Machine Learning for Intelligent Agents

N. Tziortziotis

Ph.D. Dissertation

– ♦ –

Ioannina, March 2015

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
UNIVERSITY OF IOANNINA

Τεχνικές Μηχανικής Μάθησης για την Ανάπτυξη Ευφυών Πρακτόρων

Η ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης

του Τμήματος Μηχανικών Η/Υ & Πληροφορικής Εξεταστική Επιτροπή

από τον

Νικόλαο Τζιωρτζιώτη

ως μέρος των Υποχρεώσεων για τη λήψη του

ΔΙΔΑΚΤΟΡΙΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

Μάρτιος 2015

Τριμελής Συμβουλευτική Επιτροπή

- Κωνσταντίνος Μπλέκας, Επίκουρος Καθηγητής του Τμήματος Μηχανικών Η/Υ και Πληροφορικής του Πανεπιστημίου Ιωαννίνων (Επιβλέπων)
- Αριστείδης Λύκας, Καθηγητής του Τμήματος Μηχανικών Η/Υ και Πληροφορικής του Πανεπιστημίου Ιωαννίνων
- Χριστόφορος Νίκου, Αναπληρωτής Καθηγητής του Τμήματος Μηχανικών Η/Υ και Πληροφορικής του Πανεπιστημίου Ιωαννίνων

Επταμελής Εξεταστική Επιτροπή

- Κωνσταντίνος Μπλέκας, Επίκουρος Καθηγητής του Τμήματος Μηχανικών Η/Υ και Πληροφορικής του Πανεπιστημίου Ιωαννίνων (Επιβλέπων)
- Αριστείδης Λύκας, Καθηγητής του Τμήματος Μηχανικών Η/Υ και Πληροφορικής του Πανεπιστημίου Ιωαννίνων
- Χριστόφορος Νίκου, Αναπληρωτής Καθηγητής του Τμήματος Μηχανικών Η/Υ και Πληροφορικής του Πανεπιστημίου Ιωαννίνων
- Κωνσταντίνος Βλάχος, Επίκουρος Καθηγητής του Τμήματος Μηχανικών Η/Υ και Πληροφορικής του Πανεπιστημίου Ιωαννίνων
- Γεώργιος Βούρος, Καθηγητής του Τμήματος Ψηφιακών Συστημάτων του Πανεπιστημίου Πειραιώς
- Μιχαήλ Λαγουδάκης, Αναπληρωτής Καθηγητής της Σχολής Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών του Πολυτεχνείου Κρήτης
- Ανδρέας-Γεώργιος Σταφυλοπάτης, Καθηγητής της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Ε.Μ.Π

Dedication

To my family.

Acknowledgement

I would like to sincerely thank my academic advisor Prof. Konstantinos Blekas for his valuable motivation, encouragement and assistance to my research effort during the elaboration of this dissertation. For the time and effort he spent for my guidance throughout all these years, dating back to 2008. He was always available when I needed him. Also, he was the one who introduced me to the exciting area of reinforcement learning. The collaboration with him has been a pleasant and memorable experience.

Also, I would like to thank Prof. Christos Dimitrakakis for his excellent advising during my studies as an exchange Ph.D student, at the Swiss Federal Institute of Technology Lausanne. He gave me the opportunity to collaborate with him and extend my research horizons. Our conversations along with his endless passion for research will remain unforgettable. He is one of the most brilliant persons that I have ever meet. I feel really blessed to have worked with him.

Furthermore, I am also grateful to the other members of my supervising committee Prof. Aristidis Likas and Prof. Chistophoros Nikou for their suggestions and insightful remarks. I would also like to thank Prof. Michail G. Lagoudakis, Prof. Andreas-Georgios Stafylopatis, Prof. Konstantinos Vlachos, and Prof. Georgios Vouros for serving in the examination committee of my dissertation.

I would also like to thank my colleagues Georgios Papagiannis and Konstantinos Tziortziotis for the excellent collaboration and the pleasant work environment during the last two years. I hope to meet again all together.

A big thank goes to my parents Vasileios and Anastasia, as well as to my sisters Zoi and Eirini for always believing in me and unconditionally encouraging me. This dissertation would definitely not be possible without their support and sacrifices, especially during these tough years. I am feel really grateful to have them in my life.

Finally, I feel the need to express a heartfelt thank to Katerina Pandremmenou for her support and tolerance all these years. This journey would not be so delightful without her.

Ioannina, March 2015
Nikolaos V. Tziortziotis

CONTENTS

List of Figures

List of Tables

List of Algorithms

Extended Abstract in English

Nikolaos V. Tziortziotis.
PhD, Computer Science & Engineering Department, University of Ioannina, Greece. March, 2015.
Thesis Title: Machine Learning for Intelligent Agents.
Thesis Supervisor: Konstantinos Blekas.

This dissertation studies the problem of developing intelligent agents, which are able to acquire skills in an autonomous way, simulating human behaviour. An autonomous intelligent agent acts effectively in an unknown environment, directing its activity towards achieving a specific goal based on some performance measure. Through this interaction, a rich amount of information is received, which allows the agent to perceive the consequences of its actions, identify important behavioural components, and adapt its behaviour through learning. In this direction, the present dissertation concerns the development, implementation and evaluation of machine learning techniques for building intelligent agents. Three important and very challenging tasks are considered: i) approximate reinforcement learning, where the agent's policy is evaluated and improved through the approximation of the value function, ii) Bayesian reinforcement learning, where the reinforcement learning problem is modeled as a decision-theoretic problem, by placing a prior distribution over Markov Decision Processes (MDPs) that encodes the agent's belief about the true environment, and iii) Development of intelligent agents on games, which constitute a really challenging platform for developing machine learning methodologies, involving a number of issues that should be resolved, such as the appropriate choice of state representation, continuous action spaces, etc..

In the first part, we focus on the problem of value function approximation suggesting two different methodologies. Firstly, we propose the Relevance Vector Machine Temporal Difference (RVMTD) algorithm, which constitutes an advanced kernelized Bayesian methodology for model-free value function approximation, employing the RVM regression framework as a generative model. The key aspect of RVMTD is the restructure of the policy evaluation problem as a linear regression problem. An online kernel sparsification technique is adopted, rendering the RVMTD practical in large scale domains. Based on this scheme, we derive recursive low-complexity formulas for the online update of the model observations. For the estimation of the unknown model coefficients a sparse Bayesian methodology is adopted that enhances model capabilities. Secondly,

a model-based reinforcement learning algorithm is proposed, which is based on the online partitioning of the input space into clusters. As the data arrive sequentially to the learner, an online extension of the vanilla EM algorithm is used for clustering. In this way, a number of basis functions are created and updated automatically. Also, statistics are kept about the dynamics of the environment that are subsequently used for policy evaluation. Finally, the least-squares solution is used for the estimation of the unknown coefficients of the value function model.

In the second part, we address the Bayesian reinforcement learning problem proposing two advanced Bayesian algorithms. Firstly, we present the Linear Bayesian Reinforcement Learning (LBRL) algorithm showing that the system dynamics can be estimated accurately by a Bayesian linear Gaussian model, which takes into account correlations in the state features. Policies are estimated by applying approximate dynamic programming on a transition model that is sampled from the current posterior. This form of approximate Thompson sampling results in a good exploration in unknown MDPs. Secondly, the Cover Tree Bayesian Reinforcement Learning (CTBRL) algorithm is proposed which constitutes an online tree-based Bayesian approach for reinforcement learning. The main idea of CTBRL is the construction of a cover tree from the observations, which remains efficient in high dimensional spaces. In this way, we create a set of partitions of the state space. An efficient non-parametric Bayesian conditional density estimator is also introduced on the cover tree structure. This is a generalized context tree, endowed with a multivariate linear Bayesian model at each node and is used for the estimation of the dynamics of the underlying environment. Thus, taking a sample for the posterior, we obtain a piecewise linear Gaussian model of the dynamics. The main advantages of this approach are its flexibility and efficiency, rendering it suitable for reinforcement learning problems in continuous state spaces.

In the third part of this thesis, we consider the problem of developing intelligent agents in two challenging games, the Ms. PacMan and the AngryBirds. Firstly, we propose the RL-PacMan agent, which is based on an abstract but informative state space representation. The adopted representation is able to encode a game scene, giving the opportunity to our agent to distinguish different situations. For discovering a good or even optimal policy, we use the model-free SARSA(λ) reinforcement learning algorithm. In our study, we demonstrate that an efficient state representation is of central interest for the design of an intelligent agent. Finally, we propose the AngryBER agent, which is based on an efficient tree structure for representing each game screenshot. This representation has the advantage of establishing an informative feature space and modifying the task of game playing to a regression problem. A Bayesian ensemble regression framework is used for the estimation of the return of each action, where each pair of 'object material' and 'bird type' has its own regression model. After each shot, the regression model is incrementally updated, in a fully closed form. The AngryBER agent participated in the international AIBIRDS 2014 competition winning the 2^{nd} price among 12 participants.

Εκτεταμενη Περιληψη στα Ελληνικα

Νικόλαος Τζιωρτζιώτης του Βασιλείου και της Αναστασίας.
PhD, Τμήμα Μηχανικών Η/Υ & Πληροφορικής, Πανεπιστήμιο Ιωαννίνων. Μάρτιος, 2014.
Τίτλος Διατριβής : Τεχνικές Μηχανικής Μάθησης για την Ανάπτυξη Ευφυών Πρακτόρων.
Επιβλέπων : Κωνσταντίνος Μπλέκας.

Η παρούσα διατριβή πραγματεύεται το πρόβλημα της ανάπτυξης ευφυών πρακτόρων, οι οποίοι έχουν την ικανότητα να αποκτούν δεξιότητες αυτόνομα. Ένας ευφυής πράκτορας δρα σε ένα άγνωστο περιβάλλον, κατευθυνόμενος προς την επίτευξη ενός συγκεκριμένου στόχου. Μέσω της αλληλεπίδρασης του με το περιβάλλον, ο πράκτορας λαμβάνει ένα τεράστιο όγκο πληροφοριών, που του δίνει τη δυνατότητα να αντιλαμβάνεται της συνέπειες των ενεργειών του, προσαρμόζοντας ανάλογα τη συμπεριφορά του. Στο πλαίσιο αυτό, η διατριβή επικεντρώνεται στην παρουσίαση μεθόδων Μηχανικής Μάθησης για την ανάπτυξη ευφυών πρακτόρων, εστιάζοντας σε τρεις βασικούς θεματικούς άξονες: α) προσεγγιστική ενισχυτική μάθηση, όπου η πολιτική του πράκτορα εκτιμάται και βελτιώνεται μέσω της προσέγγισης της συνάρτησης αξίας (Value Function), β) Μπεϋζιανή ενισχυτική μάθηση, όπου το πρόβλημα της ενισχυτικής μάθησης μοντελοποιείται ως ένα θεωρητικό πρόβλημα απόφασης, τοποθετώντας μια εκ των προτέρων κατανομή στο σύνολο των πιθανών Μαρκοβιανών Διαδικασιών Απόφασης (ΜΔΑ), και γ) Τεχνητή Νοημοσύνη σε Παίγνια, τα οποία αποτελούν δελεαστικά προβλήματα για την ανάπτυξη και μελέτη μεθοδολογιών μηχανικής μάθησης.

Το πρώτο μέρος της διατριβής εστιάζει στο πρόβλημα της προσέγγισης της συνάρτησης αξίας, παρουσιάζοντας δυο διαφορετικές μεθοδολογίες. Αρχικά, προτείνουμε τη μέθοδο Relevance Vector Machine Temporal Difference (RVMTD), η οποία αποτελεί μια προηγμένη Μπεϋζιανή μεθοδολογία πυρήνων για την προσέγγιση της συνάρτησης αξίας, εφαρμόζοντας το μοντέλο παλινδρόμησης RVM. Η βασική ιδέα της προτεινόμενης μεθόδου είναι ο μετασχηματισμός του προβλήματος της εκτίμησης μιας πολιτικής σε ένα πρόβλημα παλινδρόμησης. Προκειμένου ο αλγόριθμος RVMTD να καταστεί εφαρμόσιμος σε προβλήματα μεγάλης κλίμακας, υιοθετήσαμε μια τεχνική αραιών πυρήνων πραγματικού χρόνου. Βασιζόμενοι στη συγκεκριμένη τεχνική, εξάγουμε αναδρομικούς κανόνες ενημέρωσης, χαμηλής πολυπλοκότητας, που επιτρέπουν την ανανέωση των παρατηρήσεων του μοντέλου μας σε πραγματικό χρόνο. Για την εκτίμηση των άγνωστων συντελεστών του μοντέλου, υιοθετήσαμε μια αραιή Μπεϋζιανή μεθοδολογία η οποία βελτιώνει την γενικευτική ικανότητα του μοντέλου. Στη συνέχεια, προτείνουμε έναν αλγόριθμο ενισχυτικής μάθησης, ο οποίος βασίζεται στο μοντέλο του περιβάλλοντος, διαχωρίζοντας σε πραγματικό χρόνο τον χώρο

εισόδου σε ομάδες (clusters). Καθώς στο πρόβλημα της ενισχυτικής μάθησης τα δεδομένα καταφθάνουν με σειριακό τρόπο, για το πρόβλημα της ομαδοποίησης χρησιμοποιήσαμε μια εκδοχή πραγματικού χρόνου του βασικού αλγορίθμου EM. Με αυτόν τον τρόπο, επιτυγχάνουμε τη αυτόματη δημιουργία και ενημέρωση ένας συνόλου συναρτήσεων βάσης, που χρησιμοποιείται στο πρόβλημα της προσέγγισης της συνάρτησης αξίας. Τέλος, για την εκτίμηση των αγνώστων παραμέτρων του μοντέλου της συνάρτησης αξίας χρησιμοποιήσαμε τη μέθοδο των ελαχίστων τετραγώνων (least-squares solution).

Το δεύτερο μέρος της διατριβής αντιμετωπίζει το πρόβλημα της Μπεϋζιανης ενισχυτικής μάθησης, όπου προτείνονται δύο καινοτόμες μεθοδολογίες. Πρώτα, παρουσιάζεται ο αλγόριθμος Linear Bayesian Reinforcement Learning (LBRL), ο οποίος θεμελιώνει την παρατήρηση ότι ένα Μπεϋζιανό γραμμικό (Γκαουσιανό) μοντέλο είναι σε θέση να προσεγγίζει με μεγάλη ακρίβεια την δυναμική του μοντέλου του περιβάλλοντος. Οι πολιτικές εκτιμώνται εφαρμόζοντας προσεγγιστικό δυναμικό προγραμματισμό (approximate dynamic programming) στο μοντέλο μετάβασης το όποιο έχει εξαχθεί από την εκ των υστέρων κατανομή. Η συγκεκριμένη προσεγγιστική τεχνική είναι γνωστή ως δειγματοληψία Thompson και προωθεί την εξερεύνηση αγνώστων περιβαλλόντων. Στη συνέχεια, προτείνεται ο αλγόριθμος Cover Tree Bayesian Reinforcement Learning (CTBRL), ο οποίος αποτελεί μια πραγματικού χρόνου Μπεϋζιανή προσέγγιση ενισχυτικής μάθησης βασιζόμενη σε μία δενδρική δομή. Η βασική ιδέα του αλγορίθμου CTBRL είναι η κατασκευή δένδρων κάλυψης (cover trees) με βάση τις παρατηρήσεις του περιβάλλοντος, τα οποία παραμένουν αποδοτικά σε χώρους υψηλής διάστασης και χρησιμοποιούνται για την εκτίμηση της δυναμικής του προς εξέταση περιβάλλοντος. Παίρνοντας ένα δείγμα από την εκ των υστέρων κατανομή, λαμβάνουμε ένα τμηματικά, γραμμικό (piecewise linear) Γκαουσιανό μοντέλο της δυναμικής του περιβάλλοντος. Όπως και στην περίπτωση του αλγορίθμου LBRL, συνδυάζουμε τη δειγματοληψία (Thompson) με τον προσεγγιστικό δυναμικό προγραμματισμό, λαμβάνοντας αποδοτικές πολιτικές σε άγνωστα περιβάλλοντα. Τα κύρια πλεονεκτήματα της συγκεκριμένης μεθόδου είναι η αποδοτικότητά της καθώς επίσης και η ευελιξία της, καθιστώντας την κατάλληλη για προβλήματα ενισχυτικής μάθησης με συνεχείς χώρους καταστάσεων.

Το τρίτο και τελευταίο μέρος της παρούσας διατριβής, επικεντρώνεται στο πρόβλημα της ανάπτυξης ευφυών πρακτόρων για δύο δελεαστικά και συνάμα υψηλών απαιτήσεων παίγνια, το Ms. PacMan και AngryBirds. Αρχικά, προτείνουμε τον πράκτορα RL-PacMan, ο οποίος βασίζεται σε μια περιγραφική και ταυτόχρονα περιεκτική αναπαράσταση του χώρου καταστάσεων. Η προτεινόμενη αναπαράσταση κωδικοποιεί την σκηνή του παιχνιδιού με τέτοιο τρόπο έτσι ώστε να δίνεται η δυνατότητα στο πράκτορα να διακρίνει και να αντιμετωπίσει διαφορετικές καταστάσεις. Για την εξεύρεση μίας καλής πολιτικής, χρησιμοποιήσαμε τον αλγόριθμο ενισχυτικής μάθησης SARSA(λ). Η συγκεκριμένη μελέτη, επιδεικνύει ότι η σχεδίαση μιας αποδοτικής αναπαράστασης είναι σημαντική για την ανάπτυξη ενός αποδοτικού πράκτορα. Τέλος, προτείνουμε τον αλγόριθμο AngryBER ο οποίος βασίζεται σε μια αποδοτική δενδρική δομή για την αναπαράσταση της σκηνής του παιχνιδιού. Η συγκεκριμένη δομή έχει το πλεονέκτημα της εξαγωγής πληροφο-

ριακών χαρακτηριστικών και μετατρέπει το πρόβλημα της επίλυσης του παιχνιδιού σε ένα πρόβλημα παλινδρόμησης. Πιο συγκεκριμένα, χρησιμοποιούμε ένα σύνολο από Μπεϋζιανούς παλινδρομητές για την πρόβλεψη της ανταμοιβής μίας ενέργειας, όπου κάθε ζεύγος 'υλικό κατασκευής αντικειμένου' και 'τύπος πουλιού' έχουν το δικό τους μοντέλο παλινδρόμησης. Μετά το πέρας κάθε βολής, το αντίστοιχο μοντέλο παλινδρόμησης ενημερώνεται επαυξητικά, σε κλειστή μορφή. Ο πράκτορας AngryBER έλαβε μέρος στον παγκόσμιο διαγωνισμό AIBIRDS 2014, τερματίζοντας στη $2^{η}$ θέση μεταξύ των 12 συμμετεχόντων.

CHAPTER 1

INTRODUCTION

Among the major scientific challenges in our era is that of decoding human intelligence and developing artificial systems, able to mimic this intelligence. One of the most impressive aspects of human intelligence is its ability to acquire skills, without the need of an explicit teacher. The idea of learning through interaction with the environment is probably the one comes to mind first when thinking about the nature of learning. Upon interacting with the environment a plethora of useful information can be produced through trial and error. The processing of this information give us the opportunity to perceive the consequences of our actions, identify important behavioural components and modify our behaviour.

Nowadays, an enormous research effort has been observed in the direction of building intelligent agents that are able to acquire skills in an autonomous way, simulating human behaviour. In Artificial Intelligence (AI) [109], an *intelligent agent* is typically an autonomous entity, which observes and acts upon a typically unknown environment, directing its activity towards achieving specific goals. The majority of the presented artificial intelligence agents are designed based on the same concept. They accept stimulus from the environment and generate actions according to the history of stimuli they have received. Their difference originates from the fact that a number of different internal structures are used for the processing of the newly arrived information that are

generated from the interaction with the environment. Numerous intelligent agents have been proposed so far in a wide range of fields such as: robotics, gaming, navigation, etc. (see Fig. 1.1 for some indicative examples).

Figure 1.1: Intelligent Agents

1.1 Machine Learning on Intelligent Agents

Machine learning plays a crucial role in the area of artificial intelligence, as the learning capability constitutes an integral part of an intelligent system. During the last decades, a variety of machine learning techniques have been used extensively for the development of advanced decision making mechanisms, which can be considered as the agent's core. Based on these mechanisms, an agent is able to process as well as to analyse the information that is received through its interaction with the environment. In this way, the agent acquires knowledge by modifying its prior behaviour according to the received feedback in order to achieve a specific goal.

Machine learning can be divided into three main subfields according to the problem under consideration: i) *supervised learning*, ii) *unsupervised learning*, iii) *reinforcement learning*. *Supervised learning* refers to the problems where the desired outputs corresponding to some input data are known in advance. Roughly speaking, it can be seen as an explicit teacher is available. In the case where the desired output consists of discrete values, the problem is known as *classification*. The objective in classification problems is to assign an instance to one of a finite set of discrete class labels. On the other hand, if the desired output consists of continuous values, the task is called *regression*. In regression, the goal is the prediction of the output value of an unknown input instance. In contrast to supervised learning, in *unsupervised learning* no knowledge about the target values is supplied. The goal in this case is the discovery

of similar groups within the data, called as *clustering*, or the determination of the data distribution within the input space, known as *density estimation*.

Reinforcement learning has been demonstrated that constitutes a suitable platform for the development of intelligent agents. In RL, in which this thesis focuses on, an agent interacts with an initially unknown environment and modifies its behaviour (policy) so as to maximize its cumulative payoffs. In this way, reinforcement learning provides a general framework for solving complex and uncertain sequential decision problems, encountered in many real-world applications (e.g Robotics). The environment is typically modeled as a *Markov Decision Process* or *Markov Decision Problem* [102], which has been studied extensively in operations research. In order to discover a good or even optimal policy, many RL algorithms are based on the estimation of a value function (functions of states or state-action pairs), by observing data generated through the interaction of the agent with the environment. In other words, the value function indicates how good is for the agent either to be in a given state or to perform a given action in a given state. In the last years, many RL algorithms [121] have been proposed, suggesting a variety of value-function estimation techniques. Since no explicit teacher signal can be obtained in RL, the estimation of value functions differs from the regression problems encountered in supervised learning. The majority of reinforcement learning algorithms concentrate on two distinct problems: i) *policy evaluation* and ii) *control problem*. The first problem refers to the evaluation of the consequences of following a fixed policy. In contrast, the latter deals with the discovery of an optimal (or near optimal) policy that maximizes the expected future reward. Moreover, reinforcement learning algorithms can be distinguished to *model-based* (indirect) and *model-free* (direct) algorithms. The first category utilizes the knowledge of the environment dynamics so as to evaluate the value function of a fixed policy or to discover an optimal policy. In this case, the agent uses its experience in order to construct a representation of the control dynamics of its environment. In the latter case, a model-free algorithm does not need any knowledge of the environment dynamics.

This thesis concerns the development, implementation and evaluation of machine learning (reinforcement learning) methodologies for intelligent agents, focusing on three important and very challenging problems, namely approximate reinforcement learning [134, 135, 136], Bayesian reinforcement learning [137, 138], and AI in games [141, 139]. In the following, we meticulously describe the reinforcement problem, along with a review of the related work. Afterwards, we present the main contributions and the layout of the thesis.

1.2 Reinforcement Learning

In Reinforcement Learning (RL) [118] an agent performs a specific task by interacting with an unknown environment. At each time step, the agent observes the environment state and decides about the action to take in that state. According to the selected

action and the dynamics of the model of the environment, a transition to a new state is performed. At the same time, a numerical reward signal is also returned by the environment, which expresses the goal of the task, reinforcing good decision making and penalizing bad decision making. A typical setting of the interaction between the agent and the environment is shown in Figure 1.2. The notion of RL is focused on gradually improving the agent's behaviour through trial and error, by maximizing the long term expected return.

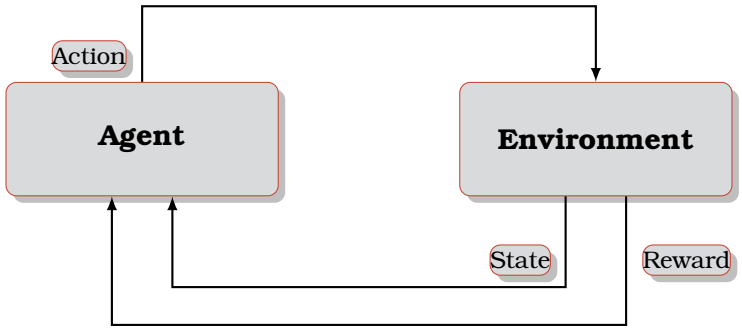


Figure 1.2: The basic reinforcement learning scenario.

In reinforcement learning, the agent's environment is modeled as a *Markov Decision Process* (MDP) which is typically denoted as a tuple $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, P, R, \gamma\}$, where:

- $\mathcal{S}$ is the state space that the agent acts in, which can be discrete or continuous.
- $\mathcal{A}$ is a non-empty set of actions, which can be discrete or continuous. In this thesis, we consider the case where a finite set of actions is available.
- $P(s'|s, a) \rightarrow [0, 1]$, is the transition probability kernel, which determines the probability of reaching state $s' \in \mathcal{S}$ from state $s \in \mathcal{S}$, by executing action a.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, is the immediate reward function, which gives the expected reward received when action $a \in \mathcal{A}$ is selected at state $s \in \mathcal{S}$.
- $\gamma \in (0, 1)$, is a discount factor such that rewards further into the future are less important than immediate rewards.

At each time step, the agent selects its actions according to a *stationary* policy $\pi \in \Pi$, which defines a conditional distribution over the actions, $\mathbb{P}^{\pi}(a \in \mathcal{A}|s)$. Roughly speaking, a *stationary* deterministic policy is a mapping $\pi : \mathcal{S} \rightarrow \mathcal{A}$ from states to actions, where $\pi(s)$ denotes the action taken in state s by the agent. A *stationary* stochastic policy can be supposed as a mapping $\pi : \mathcal{S} \rightarrow P(\mathcal{A})$ from states to action selection probabilities. The agent's utility is the *discounted return*, which is defined as the discounted sum of future rewards incurred by starting from a state s and following a policy π:

$$D^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t r_t, \tag{1.2.1}$$

where $r_t \in \mathbb{R}$ is the immediate reward received following π at step t. The objective of an RL agent is to discover a good or even *optimal policy* that achieves to maximize its expected utility over all states. In the following, the notion of value function is introduced.

1.2.1 Value Functions

The notion of value function is of central interest in RL tasks, as the majority of reinforcement learning schemes are based on estimating value functions. Value functions can be supposed as a measure of worth of a given policy π, which can be used for discovering good policies. Value functions are divided into two types: the state-value functions and the action-value functions. Given a policy π, the state-value function $V^\pi : \mathcal{S} \rightarrow \mathbb{R}$ of a state s is defined as the expected return, obtained by starting from state s and following π. Thereafter:

$$V^\pi(s) = \mathbb{E}^\pi \left\{ \sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s \right\}. \tag{1.2.2}$$

In the case where the reward function and the transition distribution are known, the Eq. 1.2.2 can be written in the following form:

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^\pi(s'), \tag{1.2.3}$$

which is known as the *Bellman equation* [13]. Thus, Eq. 1.2.3 expresses the relationship between the value of a state and the values of its successor states. Similar to the state-value function V^π, the action-value function $Q^\pi : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ expresses the expected return as received by taking action a in state s, and following policy π thereafter. Consequently:

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}^\pi \left\{ \sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a \right\}, \\ &= R(s, a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^\pi(s'). \end{aligned} \tag{1.2.4}$$

Given the state-value function V^π of a policy π, a *greedy* policy π' over π can be obtained as:

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

Similarly, if the action-value function Q^π of a policy π is known, the *greedy* policy is defined as:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}} Q^\pi(s, a).$$

A policy π' is supposed to be better than or equal to policy π, if its expected return is at least as much as that of π over all states. This can be summarized as:

$$V^{\pi'}(\boldsymbol{s}) \geq V^{\pi}(\boldsymbol{s}) \quad \text{and} \quad Q^{\pi'}(\boldsymbol{s}, \pi'(\boldsymbol{s})) \geq Q^{\pi}(\boldsymbol{s}, \pi(\boldsymbol{s})), \quad \forall \boldsymbol{s} \in \mathcal{S}. \tag{1.2.5}$$

At every MDP μ, there exists at least one policy, which is better than or equal to all other possible policies. These policies are called *optimal* and are denoted by π^*. The optimal policies share the same state-value function and action-value function, which are referred to as optimal state-value function, V^*, and optimal action-value function, Q^*, respectively.

1.2.2 Value Function Approximation

The majority of reinforcement learning algorithms rely on the estimation of a value function, which is a real-valued function over the state or the state-action space. In finite state spaces, value functions can be represented exactly using a tabular form that directly stores in memory a separate value for each individual state. Nevertheless, in the case where the state space is large or infinite (commonly encountered in the real world) an exact value representation becomes prohibitive. This problem not only stems from memory constraints, but also from the time as well as the samples needed for accurately learning all table entries. Therefore, an approximation architecture for the representation of the value function is commonly adopted, which must facilitate generalization. The most typical approximation scheme is the linear function approximation, where the value function is represented as the weighted combination of a set of basis functions:

$$V(\boldsymbol{s}) = \boldsymbol{\phi}(\boldsymbol{s})^{\top}\mathbf{w} = \sum_{i=1}^{k} \phi_i(\boldsymbol{s}) w_i, \tag{1.2.6}$$

where $\mathbf{w} \in \mathbb{R}^k$ is a vector of coefficients and $\boldsymbol{\phi} : \mathcal{S} \to \mathbb{R}^k$ is a mapping from states to a k-dimensional vector, called *basis function*. Usually, basis functions are fixed and nonlinear functions of $\boldsymbol{s}$. In this way, the value function $V(\boldsymbol{s})$, is allowed to be nonlinear function of the state space. Functions of the form of (1.2.6) are called linear models, as they are linear in the parameters $\mathbf{w}$. Nevertheless, poor design choices can result in estimates that diverge from the optimal value function and agents that perform poorly. In practice, achieving high performance requires finding an appropriate representation for the value function approximator. Next, we consider a number of approaches suitable for state space representation, which have been used extensively in the area of reinforcement learning.

State Aggregation

State aggregation is the simplest method for defining features for a linear function approximator. This approach discretizes the continuous state space into disjoint segments, whose union covers the state space $\mathcal{S}$. In this way, a binary feature is attached

to each region, which can be seen as its indicator. A feature is active (i.e. equal to 1) if the considered state falls into the corresponding region. Otherwise, the feature is 0 and is supposed as inactive.

Tile Coding (CMAC)

The idea behind tile coding [3] is the use of multiple non-overlapping partitions of the state space, known as *tilings*. Each element of a tiling, called as *tile*, is a binary feature activated if and only if the relevant state falls inside the region delineated by that tile. Typically, all the tilings are partitioned in the same way but with a slight offset from each other. Nevertheless, as suggested in [121], choosing different offsets of the tilings that corresponds to different dimensions, renders the tile coding an effective function approximation method. Moreover, an adaptive tile coding scheme is presented in [151], which begins with a simple representation with few tiles and refines it during learning by splitting existing tiles into smaller ones. Figure 1.3 illustrates a tile-coding paradigm with two tilings.

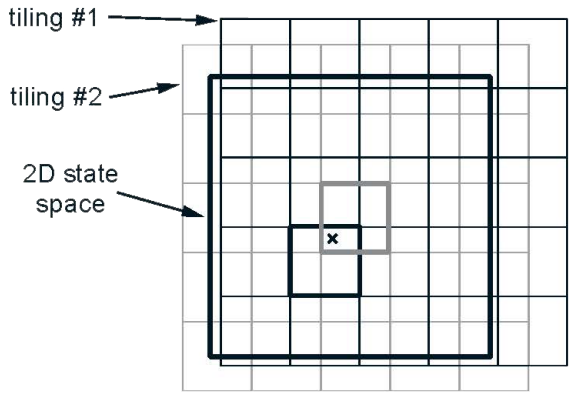


Figure 1.3: An example of tile coding with two tilings.

Radial Basis Functions

Radial Basis Functions (RBFs) are the most common basis schemes for function approximation. Unlike the other two schemes, RBFs are continuous-valued, so the feature values range in the interval $[0, 1]$. In this way, it reflects the various degrees to which the feature is present in each state. RBFs are typically localized Gaussian functions computed by:

$$\phi_i(\boldsymbol{s}) = \exp\left\{-\frac{(\boldsymbol{s} - c_i)^2}{2\sigma_i^2}\right\}, \tag{1.2.7}$$

where c_i is the center of the RBF, and σ_i^2 is the variance that determines its width. Thus, it becomes apparent that the feature value depends on the distance of the state s from the center, and the width of the RBF. Figure 1.4 shows an example with three RBFs with different centers and same width over a 1-dimensional state space.

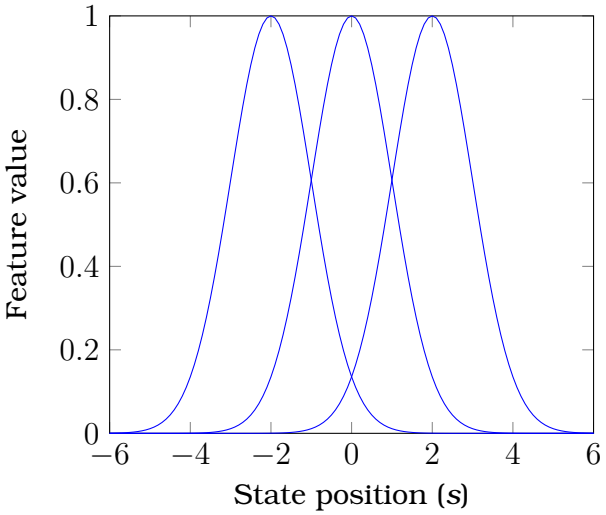


Figure 1.4: Radial basis functions in one dimension.

Usually, the centers of the RBFs are typically distributed evenly (n) along each dimension (d), producing n^d basis functions. Moreover, the width can vary, specifying the degree of function smoothness. The main advantage of RBFs over the binary features presented earlier in this section, is their ability to produce smooth and differentiable functions. On the contrary, their main drawback is the need for tuning in order for the learning to be robust and efficient.

1.3 Policy Evaluation Problem

In this section, we consider the problem of estimating the value function V^π, underlying a policy π. Policy evaluation constitutes an integral part of several control algorithms, e.g. policy iteration [102], where its target is to discover an optimal policy. In the following, we briefly review two well-known categories of algorithms for the value function estimation problem: the temporal difference learning and the least-squares temporal difference.

1.3.1 Temporal Difference Learning

The Temporal Difference (TD) family of algorithms [119] provides an elegant framework for solving prediction problems. The main advantage of this class of algorithms is its ability to learn directly from raw experience, without any further information, such as the model of the environment (*model free*). The temporal difference is a *bootstrapping* technique, where its estimates are updated online based in part on the previously learned value function estimations. More specifically, at each time t, where the agent executes the action a_t at state s_t, the predicted state-value of the newly visited state s_{t+1} along with the immediate received reward are used, in order to estimate the prediction

error, known as the *temporal difference error*:

$$\begin{aligned}\delta_t &= r_t + \gamma V_t^{\pi}(\boldsymbol{s}_{t+1}) - V_t^{\pi}(\boldsymbol{s}_t) \\ &= r_t + \gamma \boldsymbol{\phi}(\boldsymbol{s}_{t+1})^{\top}\mathbf{w}_t - \boldsymbol{\phi}(\boldsymbol{s}_t)^{\top}\mathbf{w}_t,\end{aligned} \tag{1.3.1}$$

assuming a linear approximation architecture. The *temporal difference error* is then used for adjusting the weights $\mathbf{w}_t$ of the policy π, by using a stochastic gradient descent scheme:

$$\begin{aligned}\mathbf{w}_{t+1} &= \mathbf{w}_t + \eta_t \delta_t \nabla_{\mathbf{w}_t} V_t^{\pi}(\boldsymbol{s}_t) \\ &= \mathbf{w}_t + \eta_t \delta_t \boldsymbol{\phi}(\boldsymbol{s}_t),\end{aligned} \tag{1.3.2}$$

where the parameter $\eta_t > 0$ is called the *learning rate* and controls the update rule. The above procedure is performed online, after the execution of an action by the agent. Initially, the unknown model parameters ($\mathbf{w}_0$) are set to arbitrary values, while the learning rate is set to a large value $\eta_0 \leq 1$. In fact, gradient descent methods assume that the learning rate parameter decreases over time. It is guaranteed that the temporal difference converges to the true value function of policy π, in the case where the learning rate sequence satisfies the *Robbins-Monro* conditions:

$$\sum_{t=0}^{\infty} \eta_t = \infty \quad \text{and} \quad \sum_{t=0}^{\infty} \eta_t^2 < \infty. \tag{1.3.3}$$

An extension of the classical temporal difference algorithm is the TD(λ) family of algorithms [150], where the *eligibility traces* notion is adopted. In this case, a vector of eligibility traces, $\boldsymbol{\epsilon}_t$, is maintained that indicates the extend that TD error propagates backward over the visited trajectory of states. The eligibility traces are updated at each step t, according to the next update rule:

$$\boldsymbol{\epsilon}_{t+1} = \gamma\lambda\boldsymbol{\epsilon}_t + \boldsymbol{\phi}(\boldsymbol{s}_t), \tag{1.3.4}$$

where the parameter $\lambda \in [0, 1]$ is known as *trace-decay parameter*. Initially, we set $\boldsymbol{\epsilon}_0$ equal to $\mathbf{0}$. In this way, the update rule for the model coefficients (Eq. 1.3.2) becomes:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta\delta_t\boldsymbol{\epsilon}_t. \tag{1.3.5}$$

In the case where $\lambda = 1$, the TD(1) algorithm becomes equivalent to the Monte-Carlo method [118]. On the other hand, if $\lambda = 0$, we get TD(0) that is the standard temporal difference algorithm. Therefore, it becomes apparent that the *trace-decay parameter* controls the interpolation between the Monte-Carlo and the one-step TD algorithms. Finally, in the case where $\lambda > 0$, the TD(λ) can be supposed as a multi-step method that accelerates the learning process.

1.3.2 Least Squares Temporal Difference Learning

A related class of methods is that of Least Squares Temporal Difference (LSTD) [21], which provides an appropriate way for approximating the state-value function of a fixed policy. Instead of performing gradient descent such in case of TD algorithm, the LSTD approximates the state-value function, finding the fixed-point solution in the projection space for the state-value function V:

$$\mathbf{w} = \arg\min_{\mathbf{u}\in\mathbb{R}^k} \frac{1}{2}\|\Phi^\top \mathbf{u} - (R + \gamma P^\pi \Phi^\top \mathbf{u})\|_2^2. \tag{1.3.6}$$

Since the transition matrix P^π as well as the reward function R are usually unknown, Eq.1.3.6 cannot be resolved straightforwardly. Instead, a batch of transition samples $\{(\boldsymbol{s}_i, a_i, r_i, \boldsymbol{s}'_i)\}_{i=1}^n$, is assumed, generated from the MDP of interest. Defining the sample matrices as follows:

$$\tilde{\Phi}^\top = \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{s}_1)^\top \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{s}_n)^\top \end{pmatrix}, \tilde{\Phi}'^\top = \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{s}'_1)^\top \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{s}'_n)^\top \end{pmatrix}, \tilde{R} = \begin{pmatrix} R(\boldsymbol{s}_1, a_1) \\ \vdots \\ R(\boldsymbol{s}_n, a_n) \end{pmatrix},$$

the approximated version of Eq. 1.3.6 is given by:

$$\begin{aligned} \mathbf{w} &= \arg\min_{\mathbf{u}\in\mathbb{R}^k} \frac{1}{2}\|\tilde{\Phi}^\top \mathbf{u} - (\tilde{R} + \gamma \tilde{\Phi}'^\top \mathbf{u})\|_2^2 \\ &= \arg\min_{\mathbf{u}\in\mathbb{R}^k} \frac{1}{2}\sum_{i=1}^n \{\boldsymbol{\phi}(\boldsymbol{s}_i)^\top \mathbf{u} - (R(\boldsymbol{s}_i, a_i) + \gamma \boldsymbol{\phi}(\boldsymbol{s}'_i)^\top \mathbf{u})\}^2. \end{aligned} \tag{1.3.7}$$

As it is presented in [20], the *fixed-point* solution of Eq. 1.3.7 is given by:

$$\mathbf{w} = A^{-1} b, \tag{1.3.8}$$

where

$$A^{-1} = \tilde{\Phi}(\tilde{\Phi} - \gamma \tilde{\Phi}')^\top,$$

and

$$b = \tilde{\Phi}\tilde{R}. \tag{1.3.9}$$

Thus, A is a $k \times k$ matrix and b is a vector of size $k \times 1$. It has been shown that LSTD achieves to make more efficient use of data as opposed to the vanilla TD algorithm [119]. At the same time, it does not require manual tuning of the learning rate. At this point, it should be mentioned that there exists a trade-off between the number of basis functions and the number of samples, as a large number of basis function in combination with a small number of collected samples unavoidably leads to overfitting. Furthermore, in the case where the number of basis function is extremely large, the computational demands as well as the memory requirements become unaffordable, since the storage and the inversion of matrix A is required. To overcome the specific handicaps, various regularization schemes have been proposed [72, 59] until now, trying to find an appropriate number of features, by adopting techniques from the supervised learning literature.

1.4 Control Problem

In this section, we focus our analysis on the control problem that can be considered as the complete reinforcement learning problem. In the specific problem, the objective is the discovery of optimal (or near-optimal) policies, which give the agent the opportunity to move safely inside the world, achieving its target. In the following, we present four different learning schemes that are able to tackle the control learning problem: dynamic programming, Q-learning, least-squares policy iteration, and Gaussian process reinforcement learning.

1.4.1 Dynamic Programming

Dynamic Programming (DP) refers to a class of algorithms that can be used for solving control problems. They are based on the assumption that a complete model of the environment is known in advance, for the purpose of computing optimal policies. The particular assumption as well as their computational expense (*curse of dimensionality*), render them impractical or of limited applicability. Despite these shortcomings, DP algorithms have inspired the development of more advanced reinforcement learning schemes, such as LSPI [75]. In the following of this section, we assume that the environment is a finite MDP (i.e. finite state and action spaces).

Value Iteration

Value iteration is an iterative procedure which discovers an optimal policy π^* by finding the optimal value function. Firstly, it computes the optimal value function V^*, by iteratively applying the non-linear *Bellman* equation:

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

and then, after convergence, it retrieves a policy greedily, based on the output value function V^*:

$$\pi^*(\boldsymbol{s}) = \arg\max_{a \in \mathcal{A}} \left\{ R(\boldsymbol{s}, a) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} P(\boldsymbol{s}'|\boldsymbol{s}, a) V^{*}(\boldsymbol{s}') \right\}, \qquad \forall \boldsymbol{s} \in \mathcal{S}.$$

Nevertheless, a stopping criterion is required, in order to determine when the value iteration algorithm must be terminated. An interesting remark [102] is that when the maximum difference between two successive value functions is less than a positive value ϵ, the loss between the value of the greedy policy and that of an optimal policy can be bounded as:

$$\|V - V^*\| < 2\epsilon \frac{\gamma}{1-\gamma}.$$

It is also worth mentioning that the computational cost for each iteration of the value iteration is $O(|\mathcal{S}|^2|\mathcal{A}|)$. However, the number of required iterations may grow exponentially with the discount factor γ.

Policy Iteration

Policy iteration is a dynamic programming algorithm, which starts with an arbitrary policy and steadily improves it. It discovers the optimal policy by generating a sequence of monotonically improving policies. The policy iteration algorithm manipulates the policy directly instead of finding it via the value function, as happens in the case of value iteration. Policy iteration consists of two successive, interactive phases:

- *Policy evaluation.* In this phase, the value function of policy π is computed by solving the following set of linear Bellman equations:

$$V^{\pi}(\boldsymbol{s}) = R(\boldsymbol{s}, \pi(\boldsymbol{s})) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} P(\boldsymbol{s}'|\boldsymbol{s}, \pi(\boldsymbol{s}))V^{\pi}(\boldsymbol{s}'), \qquad \forall \boldsymbol{s} \in \mathcal{S}. \tag{1.4.1}$$

- *Policy improvement.* Having computed the value function of policy π, we get a greedily improved policy:

$$\pi(\boldsymbol{s}) = \arg\max_{a \in \mathcal{A}} \left\{ R(\boldsymbol{s}, a) + \gamma \sum_{\boldsymbol{s}' \in \mathcal{S}} P(\boldsymbol{s}'|\boldsymbol{s}, a)V^{\pi}(\boldsymbol{s}') \right\}, \qquad \forall \boldsymbol{s} \in \mathcal{S}.$$

The above two phases are executed iteratively until policy π remains unchanged. In this case, the policy iteration algorithm converges to an optimal policy π^* (for more details, see Chapter 6 of [102]). Last but not least, it is worth noting that the number of iterations is upper bounded by $|\mathcal{A}|^{|\mathcal{S}|}$, which is equal to the number of distinct policies. However, the computational cost per iteration is $O(|\mathcal{S}|^2|\mathcal{A}| + |\mathcal{S}|^3)$, which becomes prohibitive in large MDPs.

1.4.2 Q-Learning

Q-learning [149] is one of the most popular reinforcement learning algorithms used for tackling the control problem. It is an off-policy algorithm that belongs to the family of temporal difference algorithms. As a *model-free* algorithm, the action-value function is approximated instead of the state-value function, in order to discover an optimal policy. It starts with an arbitrary guess for the action-value function. Then, the learning procedure takes place in an online mode. At each time step t, the agent observes the state $\boldsymbol{s}_t$ and takes the action a_t according to a followed policy. As a consequence of its action, it transits into a new state $\boldsymbol{s}_{t+1}$, receiving an immediate reward r_t. After each transition, the action-value function is updated according to the rule:

$$Q^{\pi}(\boldsymbol{s}_t, a_t) = Q^{\pi}(\boldsymbol{s}_t, a_t) + \eta(r_t + \gamma \max_{a \in \mathcal{A}} Q^{\pi}(\boldsymbol{s}_{t+1}, a) - Q^{\pi}(\boldsymbol{s}_t, a_t)), \tag{1.4.2}$$

where $\eta \in (0, 1]$ is the *learning rate* parameter. In this way, the Q-learning algorithm approximates the optimal value function Q^* irrespectively of the followed policy. Once the optimal action-value function Q^* is sufficiently approximated, an optimal or near-optimal policy can be retrieved as the greedy policy learned over the action-value

function. Under the assumptions that all state-action pairs are visited and updated continuously and the learning rate decreases sufficiently over time, Q-learning has been shown to converge with probability 1 to the optimal action-value function Q^*. Nevertheless, this is far from true in the case where the action-value function is approximated using a linear approximation architecture. This is common in the case of large problems where the value function cannot be represented explicitly using a tabular form with one entry for each state-action pair. In these cases, the action-value function is represented with the functional form of a linear model as:

$$Q^{\pi}(\boldsymbol{s}, a) = \sum_{i=1}^{k} \phi_i(\boldsymbol{s}, a) w_i = \boldsymbol{\phi}(\boldsymbol{s}, a)^{\top} \mathbf{w}, \tag{1.4.3}$$

where $\boldsymbol{\phi}(\boldsymbol{s}, a) = [\phi_1(\boldsymbol{s}, a), \ldots, \phi_k(\boldsymbol{s}, a)]^{\top}$ is the vector of *basis functions* for the state-action pair $(\boldsymbol{s}, a)$, and $\mathbf{w}$ is the unknown model parameters. Thus, the gradient-descent Q-learning rule used for updating model parameters is performed at each time step t, and is given by:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta(r_t + \gamma \max_{a \in \mathcal{A}} \boldsymbol{\phi}(\boldsymbol{s}_{t+1}, a)^{\top} \mathbf{w}_t - \boldsymbol{\phi}(\boldsymbol{s}_t, a)^{\top} \mathbf{w}_t) \boldsymbol{\phi}(\boldsymbol{s}_t, a). \tag{1.4.4}$$

Last but not least, it is worth noting that two different methods have also been presented in the literature that combine eligibility traces and Q-learning. They are the *Watkin's* Q(λ) [150] and the *Peng's* Q(λ) [96] (for a brief description, see [118]).

1.4.3 Least Squares Policy Iteration

Least-Squares Policy Iteration (LSPI) [75] is a batch approximate policy iteration algorithm, which is known for its excellent performance in data efficiency. It adopts the approximate policy-iteration framework and uses a model-free version of LSTD, called LSTD-Q. Thus, the action-value function Q, is approximated instead of the state-value function, while action selection and policy improvement are permitted without the need of any prior knowledge of the environment dynamics. As in the case of state-value function (Eq. 1.2.6), in LSTD-Q, the action-value function is approximated using a linear architecture (Eq. 1.4.3). Due to its nature, a batch of transition samples $D = \{\boldsymbol{s}_i, a_i, r_i, \boldsymbol{s}'_i | i = 1, \ldots, n\}$, are initially collected and used at each iteration in order to evaluate the derived policies. During the policy evaluation step, the matrix A and vector b, are learned following the previously learned policy π, respectively, as follows:

$$A = \sum_{i=1}^{n} \boldsymbol{\phi}(\boldsymbol{s}_i, a_i)(\boldsymbol{\phi}(\boldsymbol{s}_i, a_i) - \gamma \boldsymbol{\phi}(\boldsymbol{s}'_i, \pi(\boldsymbol{s}'_i)))^{\top}, \text{ and} \tag{1.4.5}$$

$$b = \sum_{i=1}^{n} \boldsymbol{\phi}(\boldsymbol{s}_i, a_i) r_i. \tag{1.4.6}$$

At the policy improvement step, A and b are used in order to yield a better policy. In this way, the fixed-point solution in the projection space for the approximate action-value

function Q is obtained by:

$$\mathbf{w} = A^{-1}b. \tag{1.4.7}$$

The whole procedure is implemented iteratively, until a convergence criterion over the parameters w is satisfied. The model parameters are initialized arbitrarily or are set to 0.

The standard LSPI algorithm presented above, is an offline algorithm, since a fixed set of training samples are required in order to discover an near-optimal policy. Nevertheless, a number of approaches have been proposed for this purpose, rendering the LSPI applicable in an online mode. Recently, an online variant of the LSPI has been presented in [27], where the ϵ-greedy exploration scheme is adopted. This approach is based on an online scheme, where at each time step t an action is selected greedily, based on the estimated action-value function with probability $1 - \epsilon_t$ ($\epsilon_t \in [0, 1]$), while a uniform random exploratory action is applied with probability ϵ_t. Initially, the parameter ϵ_0 is set to a large value (e.g., $\epsilon_0 = 1$), while it decays exponentially over time with a decay rate $\epsilon_d \in (0, 1)$. In the particular scheme, policy improvement can be implemented after a number of consecutive transitions. Another interesting online approach is the one presented in [78]. In that work, the R-max exploration scheme [22] is integrated in the LSPI learning algorithm.

1.4.4 Gaussian Process Reinforcement Learning

Gaussian Processes (GPs) [104] have been extensively used in a variety of supervised learning problems, such as classification or regression. Gaussian process methodologies are based on a probabilistic generative model, allowing a Bayesian treatment of these problems. Therefore, full posterior distributions can be derived that are based on both our prior beliefs and observed data. The central idea of a Gaussian process model is the definition of a prior Gaussian distribution over functions. In this way, the inference takes place directly on the function space. This fact makes GPs less restrictive as compared to the parametric models, in terms of the hypothesis space where the learning process takes place. Finally, employing the Bayes' rule, closed-form expressions for the posterior moments can also be derived. Therefore, the difficulties associated with the iterative optimization schemes and their convergence, can be overcome.

Recently, a number of Gaussian process based schemes have been proposed in the literature [103, 46, 35] that are associated with the reinforcement learning problem. The first attempt of employing GPs in reinforcement learning is that of [103], called as Gaussian Process Reinforcement Learning (GPRL). GPRL is a model-based scheme, where two GPs are employed in order to model the dynamics of the system (*dynamic GP*) and represent the value function (*value GP*). Moreover, the kernels used in the GPRL are defined as the sum of a Gaussian kernel and a weighted delta function, $K_v = K + \sigma^2\Delta$. Furthermore, the received rewards are assumed to be noiseless. Nevertheless, the assumption that the transition model and the value function are both

Gaussians, makes the calculation of the value function in a closed form intractable:

$$V^{\pi}(\boldsymbol{s}_i) = R(\boldsymbol{s}_i, \pi(\boldsymbol{s}_i)) + \gamma \int_{\mathcal{S}} P(\boldsymbol{s}'|\boldsymbol{s}_i, \pi(\boldsymbol{s}_i)) V^{\pi}(\boldsymbol{s}') d\boldsymbol{s}'. \tag{1.4.8}$$

However, this handicap can be tackled by using the solution presented in [57]. More specifically, the integral $\int_{\mathcal{S}} P(\boldsymbol{s}'|\boldsymbol{s}_i, \pi(\boldsymbol{s}_i)) V^{\pi}(\boldsymbol{s}') d\boldsymbol{s}'$ can be approximated by using $W_i K_v^{-1} V$, where W_i is the expected next kernel values given state $\boldsymbol{s}_i$. Therefore, a Gaussian distribution for the value function of all sampled points is derived, with its mean given by:

$$V = \left(I - \gamma W K_v^{-1}\right)^{-1} R, \tag{1.4.9}$$

where W is the matrix of the expected next kernel values, with $W_{ij} = \mathbb{E}[k(\boldsymbol{s}_i, \boldsymbol{s}_j)]$. Another two related schemes are those presented in [35, 34], which also focus on a model-based predictive approach. An alternative Gaussian process based approach is that of Gaussian Process Temporal Difference (GPTD), where GPs are employed for expected utility estimation (for a detailed description, see Section 2.1).

A problem with the Gaussian process reinforcement learning approaches is that they employ the marginal distribution in the dynamic programming step, ignoring the uncertainty of the model. Moreover, the output dimensions are treated independently, posing the risk of not making good use of the data. Last but not least, GPs are computationally demanding as the computational cost grows with the number of considered samples.

1.5 Thesis Contribution

In this dissertation, we study the problem of developing machine learning algorithms for intelligent agents, focusing mainly on three different axes: i) approximate reinforcement learning, ii) Bayesian reinforcement learning, iii) AI in games. The first two parts are correlated and deal with the development of efficient reinforcement learning techniques. In the third part, we mainly focus on the development of intelligent agents for two famous games, the Ms. PacMan and the Angry Birds, which constitute challenging domains. Next, we outline the main contributions of this thesis.

In Chapter 2, we suggest a practical and efficient online Bayesian kernelized reinforcement learning scheme for the policy evaluation problem, called as RVMTD. The proposed algorithmic scheme is based on temporal difference learning for the adaptation of the value function after each time step. The main concept of our approach is the restructure of the policy evaluation problem as a linear regression problem. Nevertheless, storing information for the entire history of training instances is computationally expensive. These computational requirements become prohibitive in the case where an online computation of the value estimates is required, as it usually happens with RL algorithms. To overcome this handicap, an online kernel sparsification technique [45] (i.e., approximate linear dependence) has been employed, rendering the proposed

scheme applicable in large and infinite state spaces. This technique is based on the incremental construction of a dictionary that maintains the most representative states. In this way, we are able to approximate sufficiently the images of all observed states, by combining the images of the dictionary states. Moreover, this allows us to derive recursive formulas for updating the observations of our model after each transition. Having updated the observations of our model, a sparse Bayesian regression methodology is used for the estimation of the unknown model parameters. This fact enforces sparsity leading to more general inference solutions. Last but not least, we propose an extension of the RVMTD algorithm for dealing with the control problem that allows the selection of actions and the gradual improvement of policies, without requiring knowledge of the model of the environment.

In a nutshell, we make the following contributions:

- we convert policy evaluation into a linear regression problem,
- we employ an online kernel sparsification methodology, rendering our approach practical in continuous state spaces,
- we derive recursive formulas, able to update the model at each transition with low computational cost,
- we improve the generalization capabilities of our scheme, by adopting a sparse Bayesian regression methodology,
- we present an extension of the RVMTD algorithm for learning over state-action value functions, allowing us to perform model-free policy improvement,
- we experimentally evaluate the effectiveness of our approaches on two continuous domains, namely Mountain Car and Pendulum.

In Chapter 3, we develop a model-based reinforcement learning algorithm for value function approximation in unknown environments. The presented scheme is based on the partitioning of the state space to clusters that encompass similar states. For this purpose, we adopt an appropriate mixture model, which is updated incrementally by employing an online version of the standard EM-algorithm [36]. Thus, a number of clusters are automatically constructed and updated through the interaction with the environment. In this way, a number of basis functions are also constructed based on the created clusters, and are used for the approximation of the value function, where the value function is formulated as a linear model. A number of statistics are also kept about the dynamics of the environment and are used to the policy evaluation problem. The least-squares solution is used for the estimation of the unknown coefficients of the value function model. In this way, the proposed scheme is able to estimate and update the policy followed by the agent at each time step.

In a nutshell, we make the following contributions:

- we adopt an online version of the standard EM-algorithm for online clustering,
- we extract the basis functions based on the created clusters,
- we learn the approximate dynamics of the environment by keeping statistics (i.e., transition probabilities, expected return) for each cluster,
- we approximate the value function by using the least-squares solution,
- we experimentally evaluate the performance of our approach on both simulated and real-world environments.

In Chapter 4, we propose a simple linear Bayesian approach, called as LBRL, to reinforcement learning for arbitrary state spaces. Firstly, we demonstrate that using a Bayesian linear Gaussian model is adequate for estimating the system dynamics with high accuracy. Unlike Gaussian process models, this model easily takes into account correlations in the state features, further reducing sample complexity. Another characteristic of this model is that Bayesian inference can be derived in a fully closed form. In this way, given a set of example trajectories, it is easy to sample a model from the posterior distribution. Policies are estimated by first sampling a transition model from the current posterior, and then performing Approximate Dynamic Programming (ADP) on the sampled MDP. This form of *Thompson* sampling is known to be a very efficient exploration method in bandit and discrete problems. In this work, we also demonstrate its effectiveness for continuous domains. We experiment with two different ADP approaches for finding a policy, which are Approximate Policy Iteration (API) schemes. More specifically, at the policy evaluation step, we experiment with the Fitted Value Iteration (FVI) and the Least-squares Temporal Difference (LSTD) algorithms for finding policies.

In a nutshell, we make the following contributions:

- we investigate the use of Bayesian inference under the assumption that the dynamics are linear,
- we demonstrate that a Bayesian linear Gaussian model is sufficient for accurately estimating the system dynamics,
- we show that Bayesian inference in our model is of fully closed form,
- we perform Thompson sampling that allows us to perform efficient exploration,
- we adopt the ADP scheme for the calculation of the optimal policy, by using trajectories drawn from the sampled model,
- we experimentally evaluate the effectiveness of LBRL algorithm in both offline and online modes.

In Chapter 5, we propose an online tree-based Bayesian approach for reinforcement learning, called as CTBRL. Our approach is based upon three main ideas. The first idea is to employ a cover tree [16] to create a set of partitions of the state space. This avoids having to prespecify a structure for the tree. The second technical novelty is the introduction of an efficient non-parametric Bayesian conditional density estimator on the cover tree structure. This is a generalized context tree, endowed with a multivariate linear Bayesian model at each node. This is used for estimating the dynamics of the underlying environment. The multivariate models allow for a sample-efficient estimation, by capturing dependencies. Finally, we take a sample from the posterior to obtain a piecewise linear Gaussian model of the dynamics, which can be used to generate policies. In particular, from this sample model, we obtain trajectories of simulated experience for performing ADP in order to select a policy. Although other methods could also be used to calculate optimal actions, we leave this issue for future study. The main advantages of our approach are its generality and efficiency. The posterior calculation and prediction is fully conjugate and can be performed online. At the t-th time step, inference takes $O(\ln t)$ time. Sampling from the tree, which needs only to be done infrequently, is $O(t)$. These properties are in contrast to other non-parametric approaches for reinforcement learning, such as GPs. The most computationally heavy step of our algorithm is ADP. However, once a policy is calculated, the actions to be taken can be calculated in logarithmic time at each step. The specific ADP algorithm used is not integral to our approach, and for some problems it might be more efficient to use an online algorithm.

In a nutshell, we make the following contributions:

- we employ a cover tree to create a set of partitions of the state space,
- we introduce an efficient nonparametric Bayesian conditional density estimator on the cover tree structure,
- we use a multivariate linear Bayesian model at each tree node, confirming that multivariate models are efficient for sample estimation by capturing dependencies,
- we obtain a piecewise linear Gaussian model over the system dynamics by taking a sample from the posterior,
- we perform ADP in order to obtain effective exploration policies in unknown environments, by using a number of trajectories of the sampled MDP,
- we experimentally show that cover trees are more efficient in terms of both computational cost and reward, in relation with GP models.

In Chapter 6, we deal with the problem of developing an intelligent agent, which is applied to the classical Ms. PacMan game. Reinforcement learning algorithms constitute promising methods for designing intelligent agents in games. Although their

capability of learning in real time has been already proved, the high dimensionality of state spaces in most game domains can be seen as a significant barrier. In this direction, we focus on designing of an appropriate state space representation, which constitutes the basis for the development of our agent. More specifically, we propose an abstract and at the same time informative representation of the game's state space, which incorporates all necessary information of a game snapshot. The specific representation allows our agent to distinguish different game situations, while at the same time it reduces the computational complexity, accelerates the learning procedure and enhances the generalization capability of our agent. Moreover, the on-policy reinforcement learning algorithm SARSA(λ) [108], has been used for learning a good policy. Due to the fact that the proposed state space representation is finite, the value function can be explicitly represented.

In a nutshell, we make the following contributions:

- we design an efficient state space representation,
- we use the online SARSA(λ) reinforcement learning algorithm for discovering policies,
- we demonstrate that an informative state space description plays a key role in the design of efficient RL agents,
- we perform experiments in Ms. Pac-Man domain in order to evaluate the ability of our agent to reach optimal policies.

Finally, in Chapter 7, we focus on the development of a low-complexity intelligent agent for the Angry Birds game, called as AngryBER agent. The main novelty of the proposed agent is its ability to encode the game scenes in an efficient way, by using an informative tree structure. This approach constitutes the fundamental component of the AngryBER, as it allows for the construction of an efficient and simultaneously powerful feature space that can be used during the prediction process. Moreover, based on the tree structure, we can examine if an object of the scene is directly reachable or not. In this way, we end up with a reduced set of reachable tree nodes, which are considered as possible targets. The second contribution of the proposed framework is the integration of a Bayesian ensemble regression model, where every possible pair of 'object material' - 'bird type' has its own Bayesian linear regression model. In this way, we are able to distinguish possible associations between different types of birds and object materials, improving the prediction accuracy of our scheme. Moreover, we translate the selection mechanism into a multi-armed bandit problem, where each arm corresponds to a specific regressor. The Upper Confidence Bound (UCB) algorithm [7] is adopted in this case, offering a balance between exploration and exploitation during the learning process. Finally, at the end of each shot, an online learning procedure is executed in order to adjust the model parameters of the selected regressor.

In a nutshell, we make the following contributions:

- we introduce a novel tree-like structure for encoding game scenes,
- we derive a number of features based on the tree structure,
- we propose a Bayesian ensemble regression framework for the estimation of the expected return of each taken action,
- we use an online learning scheme for updating the regression model parameters,
- we apply the UCB algorithm as a decision making mechanism, enhancing the exploration capabilities of our scheme,
- we execute experiments on several challenging game levels.

It is worth noting that the AngryBER agent participated on the 2014 AIBIRDS competition managing to win the second (2^{nd}) place among 12 participants.

1.6 Thesis Layout

The rest of this thesis is organised as follows. In Chapter 2, we propose an online Bayesian kernelized reinforcement learning scheme for the policy evaluation problem. A model-based value function approximation is presented in Chapter 3, which is based on an online clustering scheme. In Chapters 4 and 5, the Bayesian reinforcement learning problem is considered, where we propose a simple linear scheme and an online tree-based Bayesian approach for reinforcement learning, respectively. In Chapter 6, we introduce a reinforcement learning agent for the arcade Ms. Pac-Man game. Chapter 7 presents an advanced intelligent agent for playing the Angry Birds game, based on an ensemble of Bayesian regression models. Finally, Chapter 8 summarizes this dissertation and overviews directions for future work.

Chapter 2

Value Function Approximation through Sparse Bayesian Modeling

Environments with large state spaces that are encountered in real world constitute an attractive challenge for reinforcement learning. During the last years an enormous research effort has been pointed out in that direction, where the main objective is the construction of efficient approximate representations for the agent's utility function. Roughly speaking, the development of an approximate representation has been considered as the only possible method, able to cope with the curse of dimensionality. Nevertheless, there are two important preconditions that have to be taken into account in order to develop an effective approximation scheme [15]. First of all, the approximation architecture must be quite rich in order to efficiently approximate the function of interest. The second prerequisite is the existence of an efficient algorithm for tuning model parameters.

A plethora of methods have been proposed in the last decades that use a variety of value-function estimation techniques [66] and employ different approximation architectures (e.g. Radial Basis Functions (RBFs), CMACs [118], Fourier series [73], etc.). Algorithms such as the Q-learning [149] and Sarsa [108, 114] try to estimate the long-term expected value of each possible action, given a particular state. Least Squares Temporal Difference (LSTD) learning [21] is a widely used algorithm for the policy evaluation problem. Also, Least Squares Policy Iteration (LSPI)[75] is an offline method,

which extends the LSTD, by using it in the evaluation step of the approximate policy iteration algorithm.

Recently, kernelized reinforcement learning methods have gain a lot of attention by employing all the benefits of kernel techniques [124]. One of the earliest works in that direction is the well-known Kernel-Based Reinforcement Learning (KBRL) algorithm [90], where the value function is approximated offline by using a kernel-based locally weighted averaging. In [63] the authors proposed a combination of Prioritized Sweeping, which employs model-based exploration, with kernel based function approximation. Furthermore, standard RL methods (e.g., LSTD, LSPI, etc.) have been extended to kernel spaces [156, 155]. Gaussian Processes (GPs) also have been recently used in reinforcement learning offering an elegant Bayesian RL formulation. Gaussian Processes in Reinforcement Learning (GPRL) [103] is a model-based RL algorithm, where two different Gaussian Processes are used sequentially. The first one is used to learn the transition model of the environment, while the second one is used to approximate the value function. Gaussian Process Temporal Difference (GPTD) [46] constitutes an alternative adaptation of the Gaussian processes to the problem of online value-function estimation. GPTD employs a probabilistic generative model over the state value function, while the solution to the inference problem is given by the posterior distribution, conditioned on the observed sequence of received rewards. It incrementally constructs an appropriate dictionary of representative states. In [107], a classifier-based policy search technique has been proposed, where each action is viewed as a distinct class and the states are the instances to be classified. The Relevance Vector Machine (RVM) has been employed so as to identify the critical parts of the state space. Finally, the Kalman Temporal Difference (KTD) framework has been introduced in [55], where the value function approximation is formulated as a filtering problem and non-stationary systems are allowed.

In this chapter, the Relevance Vector Machine Temporal Difference (RVMTD) algorithm is presented, which constitutes an online Bayesian kernelized reinforcement learning scheme for the policy evaluation problem. The key aspect of RVMTD is the restructure of the policy evaluation problem as a linear regression problem. In order to render the RVMTD algorithm practical in large scale domains, an online sparsification kernel technique has been adopted [45]. The specific technique is based on an incremental construction of a dictionary that holds the most representative states. Therefore, the feature images of all observed states can be sufficiently approximated, by combining the images of the dictionary states. The advantages of this approach are summarized as follows. Firstly, we achieve a reduced computational complexity, since our analysis deals only with a small fraction of the encountered states. Secondly, the automatic feature space selection holds only most significant states. Thirdly, sparsity is encouraged, which improves the generalization ability of our solutions. Finally, we derive recursive formulas, able to update our model (e.g. model observations) at each transition. The learning of the regression model parameters is performed through

a sparse Bayesian methodology [129, 112] that offers many advantages in regression problem. Enforcing sparsity is a fundamental machine learning regularization principle that leads to more flexible inference solutions. In sparse Bayesian regression, we employ models that initially have many degrees of freedom, and a heavy tail prior is applied over the coefficients. At the end of the learning process, only few of the them are automatically maintained, as most significant. This approach is equivalent to retaining only a part of the dictionary states, which are responsible for approximating the value function and designing the optimum policy. Furthermore, an incremental computationally-efficient optimization strategy has also been adopted [130], in order to accelerate the learning procedure of the model parameters. Finally, we have presented an extension of the RVMTD algorithm, where the action-value function is learned instead of state-value function. This fact allows the proposed scheme to tackle the complete reinforcement learning problem, by applying model-free policy improvement so as to discover optimal or near-optimal policies. Comparisons with the GPTD algorithm have been conducted in two well-known benchmark problems, showing that RVMTD significantly outperforms the other approach.

The rest of this chapter is organized as follows. In Section 2.1, we briefly describe the GPTD methodology as a Bayesian framework for value function approximation. The proposed sparse regression model is presented in Section 2.2, along with an incremental learning optimization scheme used for value function estimation. Numerical experiments are presented in Section 2.3 in order to assess the performance of our methodology. Finally, Section 2.4 concludes this chapter.

2.1 Gaussian Process Temporal Difference

Gaussian Process Temporal Difference (GPTD) learning algorithm offers a Bayesian solution to the policy evaluation problem. A probabilistic generative model is used for the value function, by imposing a Gaussian prior over value functions and assuming Gaussian noise. Thus, Bayesian reasoning can be applied to the value estimation problem, of a policy π, when the MDP μ is unknown. In this way, the solution to the inference problem is given by the posterior distribution of the value function, conditioned on the observed sequence of rewards, received after following policy π, which is also Gaussian and is described by its mean and covariance. In the following, we concisely present the basic GPTD model (see [46] for a more detailed overview).

The basis of the GPTD statistical generative model is the decomposition of the agent's utility (i.e., discounted return) into its mean value and a random zero mean residual ΔV:

$$D^{\pi}(\boldsymbol{s}) = V^{\pi}(\boldsymbol{s}) + (D^{\pi}(\boldsymbol{s}) - V^{\pi}(\boldsymbol{s})) = V^{\pi}(\boldsymbol{s}) + \Delta V^{\pi}(\boldsymbol{s}). \tag{2.1.1}$$

Moreover, using the stationarity of the MDP, the discounted return (Eq. 1.2.1) may be

written as:

$$D^\pi(\boldsymbol{s}) = R^\pi(\boldsymbol{s}) + \gamma D^\pi(\boldsymbol{s}'), \quad \text{with } \boldsymbol{s}' \sim P(\cdot|\boldsymbol{s}, \pi(\boldsymbol{s})). \tag{2.1.2}$$

Substituting Eq. 2.1.1 into 2.1.2, the following rule is obtained:

$$R^\pi(\boldsymbol{s}) = V^\pi(\boldsymbol{s}) - \gamma V^\pi(\boldsymbol{s}') + N^\pi(\boldsymbol{s}, \boldsymbol{s}') \ , \tag{2.1.3}$$

where $N^\pi(\boldsymbol{s}, \boldsymbol{s}') \triangleq \Delta V^\pi(\boldsymbol{s}) - \gamma \Delta V^\pi(\boldsymbol{s}')$ is the difference between residuals. Given a sample trajectory of states $\{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_t\}$, we get a system of t linear equations that may be comprehensively expressed in a matrix form as:

$$R_{t-1} = H_t V_t + N_t, \tag{2.1.4}$$

where

$$\begin{aligned} R_t &= (R^\pi(\boldsymbol{s}_1), \ldots, R^\pi(\boldsymbol{s}_t))^\top, \\ V_t &= (V^\pi(\boldsymbol{s}_1), \ldots, V^\pi(\boldsymbol{s}_t))^\top, \\ N_t &= (N(\boldsymbol{s}_1, \boldsymbol{s}_1), \ldots, N(\boldsymbol{s}_{t-1}, \boldsymbol{s}_t))\top, \end{aligned}$$

are vectors of rewards, value functions, and residuals, respectively. Also, H_t is a $(t-1) \times t$ rectangular matrix of the form:

$$H_t = \begin{bmatrix} 1 & -\gamma & 0 & \cdots & 0 \\ 0 & 1 & -\gamma & \cdots & 0 \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & \cdots & 1 & -\gamma \end{bmatrix}. \tag{2.1.5}$$

The above equation can be described as a Gaussian Process, assuming a zero-mean Gaussian distribution over the value functions, i.e., $V_t \sim \mathcal{N}(\mathbf{0}, K_t)$, where K_t is a kernel covariance matrix that reflects the similarity of states in the environment. Similarly, we assume that each one of the residuals, $\Delta V^\pi(\boldsymbol{s}_i)$, is generated independently of all the others (i.e., $[\Delta V^\pi(\boldsymbol{s}_i) \Delta V^\pi(\boldsymbol{s}_j)] = 0$, for $i \neq j$) and follows a white Gaussian distribution with variance, $\sigma_i = \sigma^2$, for all $i \in \{1, \ldots, t\}$. Since $N_t = H_t \Delta V_t$, it becomes apparent that the noise vector is zero-mean Gaussian, $N_t \sim \mathcal{N}(\mathbf{0}, \Sigma_t)$, with the covariance given as: $\Sigma_t = \sigma_t^2 H_t H_t^\top$. Thus, the posterior distribution over the value of state $\boldsymbol{s}$, conditioned on an observed sequence of rewards, is also Gaussian:

$$(V^\pi(\boldsymbol{s})|R_{t-1}) \sim \mathcal{N}(\hat{V}(\boldsymbol{s}), p_t(\boldsymbol{s})), \tag{2.1.6}$$

where mean and covariance are respectively given by:

$$\hat{V}(\boldsymbol{s}) = \mathbf{k}_t(\boldsymbol{s})^\top \mathbf{q}_t, \quad p_t(\boldsymbol{s}) = k(\boldsymbol{s}, \boldsymbol{s}) - \mathbf{k}_t(\boldsymbol{s})^\top C_t \mathbf{k}_t(\boldsymbol{s}) \tag{2.1.7}$$

with

$$\mathbf{q}_t = H_t^\top (H_t K_t H_t^\top + \Sigma_t)^{-1} R_{t-1} \quad \text{and} \quad C_t = H_t^\top (H_t K_t H_t^\top + \Sigma_t)^{-1} H_t. \tag{2.1.8}$$

Nevertheless, evaluating the parameters $\mathbf{q}_t$ and C_t of the posterior is computationally infeasible for large sets of samples, since we need to store and invert a matrix of size $t \times t$ at a cost of $O(t^3)$.

2.1.1 Online Sparcification

A key restriction on the application of kernel methods, such as Gaussian Processes, in large scale problems is that the number of unknown model coefficients is equal to the number of sampled data. Apart from reducing the generalization capability, this may lead to severe computational problems, as the computational cost is linear with the size of the kernel matrix. These problems become much more evident in reinforcement learning approaches, such as GPTD algorithm, where new input samples are arrived sequentially over time.

In this direction, an online kernel sparsification methodology has been proposed in [45], which is based on the Approximate Linear Dependency (ALD) analysis. To perform ALD analysis, a dictionary $\mathcal{D}$ with the most representative states is defined, which is initially empty, $\mathcal{D}_0 = \{\}$. At each time step t, a state $\boldsymbol{s}_t$ is observed and tested if its feature space image $\boldsymbol{\phi}(\boldsymbol{s}_t)$ can be approximated adequately, by combining the images of states already admitted in the dictionary. Thus, having observed $t-1$ training samples $\{\boldsymbol{s}_i\}_{i=1}^{t-1}$ and supposing that a dictionary consisting of a subset of the training samples, $\mathcal{D}_{t-1} = \{\tilde{\boldsymbol{s}}_1, \ldots, \tilde{\boldsymbol{s}}_{|\mathcal{D}_{t-1}|}\}$, has been created, the approximation of $\boldsymbol{\phi}(\boldsymbol{s}_t)$ in terms of this dictionary is given by the solution of the following least-squares problem:

$$\min_{\boldsymbol{\alpha}} \left\| \sum_{j=1}^{\mathcal{D}_{t-1}} \alpha_j \boldsymbol{\phi}(\boldsymbol{s}_j) - \boldsymbol{\phi}(\boldsymbol{s}_t) \right\|^2 . \tag{2.1.9}$$

According to the Mercer's Theorem, the kernel function can be viewed as an inner product in a general high dimensional *Hilbert* space $\mathcal{H}$, via a general non-linear mapping $\boldsymbol{\phi} : \mathcal{S} \rightarrow \mathcal{H}$, for which $\langle \boldsymbol{\phi}(\boldsymbol{s}), \boldsymbol{\phi}(\boldsymbol{s}') \rangle_{\mathcal{H}} = k(\boldsymbol{s}, \boldsymbol{s}')$ (see [111], for details). Thus, expanding Eq. 2.1.9 and employing the kernel trick, the following minimization problem is formulated:

$$\min_{\boldsymbol{\alpha}} \left\{ \boldsymbol{\alpha}^\top \tilde{K}_{t-1} \boldsymbol{\alpha} - 2\boldsymbol{\alpha}^\top \tilde{\mathbf{k}}_{t-1}(\boldsymbol{s}_t) + 1 \right\}, \tag{2.1.10}$$

where $\tilde{\mathbf{k}}_{t-1}(\boldsymbol{s}_t) = (k(\tilde{\boldsymbol{s}}_1, \boldsymbol{s}_t), \ldots, k(\tilde{\boldsymbol{s}}_{|\mathcal{D}_{t-1}|}, \boldsymbol{s}_t))^\top$ is a $|\mathcal{D}_{t-1}| \times 1$ vector, and $\tilde{K}_{t-1}$ is the kernel matrix of the dictionary states at time $t-1$ (i.e., $\tilde{K}_{t-1} = [\tilde{\mathbf{k}}_{t-1}(\tilde{\boldsymbol{s}}_1), \ldots, \tilde{\mathbf{k}}_{t-1}(\tilde{\boldsymbol{s}}_{|\mathcal{D}_{t-1}|})]$), which is square, symmetric, and positive definite. The solution to the above optimization problem is given by:

$$\boldsymbol{\alpha}_t = \tilde{K}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(\boldsymbol{s}_t). \tag{2.1.11}$$

Thus, the squared error incurred by the approximation is:

$$\delta_t = 1 - \tilde{\mathbf{k}}_{t-1}(\boldsymbol{s}_t)^\top \boldsymbol{\alpha}_t = 1 - \tilde{\mathbf{k}}_{t-1}(\boldsymbol{s}_t)^\top \tilde{K}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(\boldsymbol{s}_t). \tag{2.1.12}$$

If $\delta_t > \nu$, the state $\boldsymbol{s}_t$ is inserted to the dictionary, $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\boldsymbol{s}_t\}$. Now, as $\boldsymbol{s}_t$ is represented explicitly by itself, we set $\boldsymbol{\alpha}_t = (0, \ldots, 0, 1)^\top$. Otherwise, if $\delta_t \leq \nu$, the dictionary remains unchanged, $\mathcal{D}_t = \mathcal{D}_{t-1}$. It is worth noting that ν is an accuracy threshold parameter that controls the level of sparsity in the model[1]. In this way, all

[1]For $\nu = 0$, the approximations become exact.

feature vectors that correspond to the states observed up to time t can be sufficiently approximated by combining the images of the states already located in the dictionary $\mathcal{D}_t$, with a maximum squared error ν:

$$\boldsymbol{\phi}(\boldsymbol{s}_i) = \sum_{j=1}^{|\mathcal{D}_t|} \alpha_{i,j} \boldsymbol{\phi}(\tilde{\boldsymbol{s}}_j) + \boldsymbol{\phi}_i^{res}, \quad \text{where } \|\boldsymbol{\phi}_i^{res}\|^2 \leq \nu. \tag{2.1.13}$$

Therefore, for all samples encountered up to time t, Eq. 2.1.13 can be written in a more concise form as:

$$\Phi_t = \tilde{\Phi}_t A_t^\top + \Phi_t^{res} \tag{2.1.14}$$

where $A_t = [\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_t]^\top$, $\Phi_t = [\boldsymbol{\phi}(\boldsymbol{s}_1), \ldots, \boldsymbol{\phi}(\boldsymbol{s}_t)]$, $\tilde{\Phi}_t = [\boldsymbol{\phi}(\tilde{\boldsymbol{s}}_1), \ldots, \boldsymbol{\phi}(\tilde{\boldsymbol{s}}_{|\mathcal{D}_t|})]$, and $\Phi^{res} = [\boldsymbol{\phi}_1^{res}, \ldots, \boldsymbol{\phi}_t^{res}]$. Moreover, multiplying Eq. 2.1.14 by its transpose and making use of the kernel trick, we obtain:

$$K_t \approx A_t \tilde{K}_t A_t^\top. \tag{2.1.15}$$

Similarly, since $k_t(\boldsymbol{s}) = \Phi_t^\top \boldsymbol{\phi}(\boldsymbol{s})$, it holds that:

$$\mathbf{k}_t(\boldsymbol{s}) \approx A_t \tilde{\mathbf{k}}_t(\boldsymbol{s}). \tag{2.1.16}$$

2.1.2 Online Gaussian Process TD

Algorithm 2.1: Online Gaussian Process Temporal Difference

Input : ν, σ_0, γ, $\boldsymbol{s}_1$.
Output : $\mathcal{D}_t, \tilde{q}_t, \tilde{C}_t$.
Initialize: $\mathcal{D}_1 = \{\boldsymbol{s}_1\}, \tilde{q}_1 = 0, \tilde{C}_1 = 0, \tilde{K}_1^{-1} = 1$.

for $t = 2, 3, \ldots$ **do**
  Perform transition $\boldsymbol{s}_{t-1} \to \boldsymbol{s}_t$;
  Observe reward $R(\boldsymbol{s}_{t-1})$;
  if $\delta_t > \nu$ **then** (2.1.12) `// Insert` $\boldsymbol{s}_t$ `to the dictionary`
    $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\boldsymbol{s}_t\}$;
    Compute $\tilde{K}_t^{-1}$;
  end
  Compute $\tilde{q}_t$ and $\tilde{C}_t$ (2.1.18);
end

An online sparse algorithm for GPTD is derived (see [46], for details) by substituting the approximations (2.1.15, 2.1.16) to the GP solution (2.1.7), as given below:

$$\hat{V}_t(\boldsymbol{s}) = \tilde{\mathbf{k}}_t(\boldsymbol{s}) \tilde{\mathbf{q}}_t \quad \text{and} \quad p_t(\boldsymbol{s}) = 1 - \tilde{\mathbf{k}}_t(\boldsymbol{s})^\top \tilde{C}_t \tilde{\mathbf{k}}_t(\boldsymbol{s}), \tag{2.1.17}$$

where

$$\tilde{\mathbf{q}}_t = \tilde{H}_t^\top (\tilde{H}_t \tilde{K}_t \tilde{H}_t^\top + \Sigma_t)^{-1} R_{t-1}, \qquad \tilde{C}_t = \tilde{H}_t^\top (\tilde{H}_t \tilde{K}_t \tilde{H}_t^\top + \Sigma_t)^{-1} \tilde{H}_t \tag{2.1.18}$$

and $\tilde{H}_t \triangleq H_t A_t$. Thus, the only parameters that need to be stored and updated are $\tilde{\mathbf{q}}_t$ and $\tilde{C}_t$, whose dimensions are $|\mathcal{D}_t| \times 1$ and $|\mathcal{D}_t| \times |\mathcal{D}_t|$, respectively. This fact renders the GPTD algorithm practical, since it results in significant computational savings. Recursive update formulas for the above mean $\tilde{\mathbf{q}}_t$ and covariance $\tilde{C}_t$ may be also derived [46]. Algorithm 2.1 illustrates the basic steps of the online GPTD.

2.2 Relevance Vector Machine Temporal Difference

In this section, we present the Relevance Vector Machine Temporal Difference (RVMTD) methodology, which deals with the tasks of policy evaluation and control learning. The main idea of RVMTD is to convert the reinforcement learning task into a regression problem. The concept of a dictionary containing the most representative states has been adopted in order to reduce the computational complexity of the RVMTD algorithm, rendering it practical in large scale domains. Finally, an incremental, highly-accelerated version of the standard Relevance Vector Machine (RVM) [129] regression scheme has also been employed for the value function approximation.

We now explain the basic components of RVMTD algorithm in detail. Firstly, we give the general scheme of the RVMTD algorithm, transforming the RL policy evaluation problem into a regression problem. Next, in Section 2.2.2, we present the optimization RVM scheme used for tuning the model parameters. In Section 2.2.3 an episodic version of RVMTD is described, while Section 2.2.4 presents a modified version of RVMTD that is able to solve the complete RL problem (decision making), namely, the task of finding optimal or near-optimal policies in unknown environments.

2.2.1 Relevance Vector Machine TD for Policy Evaluation

Policy evaluation constitutes one of the most crucial reinforcement learning components as it constitutes an integral part of many RL algorithmic schemes including policy iteration. This section introduces a Bayesian solution to the policy evaluation problem in general state spaces, by using function approximation. Following the same analysis as that of GPTD, the sequence of rewards corresponding to a sample trajectory of states $\{\mathbf{s}_1, \ldots, \mathbf{s}_t\}$ can be expressed as:

$$R_{t-1} = H_t V_t + N_t. \tag{2.2.1}$$

In our approach, we assume that the (hidden) vector of the value functions is described with the functional form of a linear model:

$$V_t = \Phi_t^\top \mathbf{w}_t \ , \tag{2.2.2}$$

where $\mathbf{w}_t$ is the vector of the t unknown model regression coefficients and $\Phi_t = [\boldsymbol{\phi}(\mathbf{s}_1), \ldots, \boldsymbol{\phi}(\mathbf{s}_t)]$ is the 'design' matrix that contains t state basis functions. There-

fore, Eq. 2.2.1 may be written as:

$$R_{t-1} = H_t \Phi_t^\top \mathbf{w}_t + N_t. \tag{2.2.3}$$

As the feature space may be of high dimensionality, the manipulation of matrices such as Φ_t becomes prohibitive. To overcome this problem, we follow the suggestions presented in [45], expressing the weight vector $\mathbf{w}_t$ as the weighted sum of the state feature vectors:

$$\mathbf{w}_t = \sum_{i=1}^{t} \boldsymbol{\phi}(\boldsymbol{s}_t)\mathsf{w}_i = \Phi_t \mathsf{w}, \tag{2.2.4}$$

where $\mathsf{w} = (\mathsf{w}_1, \ldots, \mathsf{w}_t)^\top$ are the corresponding coefficients. Substituting Eq. 2.2.4 into Eq. 2.2.3 and using the kernel trick, we get:

$$R_{t-1} = H_t K_t \mathsf{w} + N_t. \tag{2.2.5}$$

The problem with the resulting regression scheme is that the order of the model is equal to the number of training samples, leading to severe overfitting. Moreover, as the training samples arrive sequentially, one at each time step, the estimation of the coefficient vector w and the evaluation of the value function of a new state become prohibitive in terms of both memory and time. To overcome these handicaps and make the algorithm practical reducing the computational complexity, we adopt the sparsification method described in Section 2.1.1. More specifically, we use a smaller set of samples instead of the entire training set, which becomes extremely large or infinite as mentioned above. Therefore, we maintain a dictionary of samples that is adequate to approximate any training sample in the feature space with high accuracy (i.e. the level of accuracy is determined by the positive threshold parameter ν). Using the approximation $\Phi_t \approx \tilde{\Phi}_t A^\top$ (Eq. 2.1.14) and replacing at Eq. 2.2.4, we get:

$$\mathbf{w}_t = \Phi_t \mathsf{w} \approx \tilde{\Phi}_t A_t^\top \mathsf{w} = \tilde{\Phi}_t \tilde{\mathbf{w}}_t = \sum_{j=1}^{|\mathcal{D}_t|} \tilde{w}_j \boldsymbol{\phi}(\boldsymbol{s}_j), \tag{2.2.6}$$

where $\tilde{\mathbf{w}}_t \triangleq A_t^\top \mathsf{w}$ is a $|\mathcal{D}_t| \times 1$ vector of unknown coefficients. Thus, Eq. 2.2.3 gets the following form:

$$\begin{aligned} R_{t-1} &= H_t \Phi_t \tilde{\Phi}_t \tilde{\mathbf{w}}_t + N_t \\ &= H_t A_t \tilde{K}_t \tilde{\mathbf{w}}_t + N_t \\ &= \tilde{H}_t \tilde{K}_t \tilde{\mathbf{w}}_t + N_t, \end{aligned} \tag{2.2.7}$$

where $\tilde{H}_t \triangleq H_t A_t$ and $\tilde{K}_t$ is the kernel matrix of the dictionary states at time step t (i.e., $\tilde{K}_t = [\tilde{\mathbf{k}}_t(\tilde{\mathsf{s}}_1), \ldots, \tilde{\mathbf{k}}_t(\tilde{\mathsf{s}}_{|\mathcal{D}_t|})]$) . Taking the pseudoinverse[2] of $\tilde{H}_t$, we end up using the following linear regression model:

$$\begin{aligned} \tilde{H}_t^\dagger R_{t-1} &= \tilde{K}_t \tilde{\mathbf{w}}_t + \tilde{H}_t^\dagger N_t \Rightarrow \\ (\tilde{H}_t^\top \tilde{H}_t)^{-1} \tilde{H}_t^\top R_{t-1} &= \tilde{K}_t \tilde{\mathbf{w}}_t + \tilde{H}_t^\dagger N_t \Rightarrow \\ P_t \mathbf{v}_t &= \tilde{K}_t \tilde{\mathbf{w}}_t + e_t, \end{aligned} \tag{2.2.8}$$

[2] $M^\dagger$ indicates the pseudoinverse of matrix M.

where $P_t \triangleq (\tilde{H}_t^\top \tilde{H}_t)^{-1}$, $\mathbf{v}_t \triangleq \tilde{H}_t^\top R_{t-1}$, and $e_t \triangleq \tilde{H}_t^\dagger N_t$. The vector e_t plays the role of model noise. It a nutshell, the only parameters that need to be stored and updated are P_t, $\tilde{K}_t$ and $\mathbf{v}_t$, whose dimensions are $|\mathcal{D}_t| \times |\mathcal{D}_t|$, $|\mathcal{D}_t| \times |\mathcal{D}_t|$ and $|\mathcal{D}_t| \times 1$, respectively. In the following, we derive update rules for P_t, $\tilde{K}_t$ and $\mathbf{v}_t$. At each time step t, where the agent moves to a new state $\boldsymbol{s}_t$ receiving a reward r_{t-1} , we fall in one of the following two cases:

1. $\delta_t \leq \nu$. In this case, the dictionary remains unchanged (i.e., $\mathcal{D}_t = \mathcal{D}_{t-1}$) as the feature vector $\boldsymbol{\phi}(\boldsymbol{s}_t)$ can be sufficiently approximated by combining the images of the dictionary states.

2. $\delta_t > \nu$. The state $\boldsymbol{s}_t$ is added to the dictionary (i.e., $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\boldsymbol{s}_t\}$) as the $\boldsymbol{\phi}(\boldsymbol{s}_t)$ is not approximate linear dependent on dictionary states $\mathcal{D}_{t-1}$. Therefore, the dimension of dictionary kernel $\tilde{K}_t$ increases by 1.

At this point, we derive the recursive RVMTD update equations for each one of the above two cases.

Case 1: $\mathcal{D}_t = \mathcal{D}_{t-1}$

Due to the fact that the dictionary remains unchanged, $\tilde{K}_t = \tilde{K}_{t-1}$ resulting to $\tilde{K}_t^{-1} = \tilde{K}_{t-1}^{-1}$. In the specific case, only matrices A_t and H_t change between successive time steps:

$$A_t = \begin{bmatrix} A_{t-1} \\ \tilde{\boldsymbol{\alpha}}_t^\top \end{bmatrix}, \qquad H_t = \left[\begin{array}{cc} H_{t-1} & \mathbf{0} \\ \hline \mathbf{h}_t \end{array}\right], \tag{2.2.9}$$

where $\tilde{\boldsymbol{\alpha}}_t = \tilde{K}_{t-1}^{-1} \tilde{k}_{t-1}(\boldsymbol{s}_t)$ (given by Eq. 2.1.11), $\mathbf{h}_t = (0, \ldots, 0, 1, -\gamma)^\top$ and $\mathbf{0}$ is a vector of zeros of appropriate length. Therefore,

$$\tilde{H}_t = H_t A_t = \begin{bmatrix} \tilde{H}_{t-1} \\ \tilde{\boldsymbol{\alpha}}_{t-1}^\top - \gamma \tilde{\boldsymbol{\alpha}}_t^\top \end{bmatrix} = \begin{bmatrix} \tilde{H}_{t-1} \\ \tilde{\mathbf{h}}_t^\top \end{bmatrix}, \tag{2.2.10}$$

where we define $\tilde{\mathbf{h}}_t \triangleq \tilde{\boldsymbol{\alpha}}_{t-1} - \gamma \tilde{\boldsymbol{\alpha}}_t$. Multiplying the transpose of Eq. 2.2.10 by itself, we get the following recursive formula:

$$\begin{aligned} \tilde{H}_t^\top \tilde{H}_t &= \begin{bmatrix} \tilde{H}_{t-1}^\top & \tilde{\mathbf{h}}_t \end{bmatrix} \begin{bmatrix} \tilde{H}_{t-1} \\ \tilde{\mathbf{h}}_t^\top \end{bmatrix} \\ &= \tilde{H}_{t-1}^\top \tilde{H}_{t-1} + \tilde{\mathbf{h}}_t \tilde{\mathbf{h}}_t^\top. \end{aligned} \tag{2.2.11}$$

In this way, we derive the following recursive formula for P_t, by using the matrix inversion Lemma (Appendix A.2):

$$\begin{aligned} P_t &= (\tilde{H}_{t-1}^\top \tilde{H}_{t-1} + \tilde{\mathbf{h}}_t \tilde{\mathbf{h}}_t)^{-1} \\ &= P_{t-1} - \frac{P_{t-1} \tilde{\mathbf{h}}_t \tilde{\mathbf{h}}_t^\top P_{t-1}}{1 + \tilde{\mathbf{h}}_t^\top P_{t-1} \tilde{\mathbf{h}}_t}. \end{aligned} \tag{2.2.12}$$

Finally, it is easily seen that the vector $\mathbf{v}_t$ can be obtained recursively as follows:

$$\begin{aligned}\mathbf{v}_t = \tilde{H}_t^\top R_{t-1} &= \begin{bmatrix}\tilde{H}_{t-1}^\top & \tilde{\mathbf{h}}_t\end{bmatrix}\begin{bmatrix} R_{t-2} \\ r_{t-1}\end{bmatrix} \\ &= \tilde{H}_{t-1}^\top R_{t-2} + \tilde{\mathbf{h}}_t r_{t-1} \\ &= \mathbf{v}_{t-1} + \tilde{\mathbf{h}}_t r_{t-1}. \end{aligned} \tag{2.2.13}$$

Case 2: $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\mathbf{s}_t\}$

In this case, $\tilde{K}_t \neq \tilde{K}_{t-1}$, as state $\mathbf{s}_t$ is added to the dictionary. Now, the dictionary kernel matrix is given by:

$$\tilde{K}_t = \begin{bmatrix} \tilde{K}_{t-1} & \tilde{\mathbf{k}}_{t-1}(\mathbf{s}_t) \\ \tilde{\mathbf{k}}_{t-1}(\mathbf{s}_t)^\top & 1 \end{bmatrix}. \tag{2.2.14}$$

Using the partitioned matrix inverse formula, we derive the following recursive formula for $\tilde{K}_t^{-1}$ (see Appendix A.2):

$$\tilde{K}_t^{-1} = \frac{1}{\delta_t}\begin{bmatrix} \delta_t \tilde{K}_{t-1}^{-1} + \tilde{\boldsymbol{\alpha}}_t \tilde{\boldsymbol{\alpha}}_t^\top & -\tilde{\boldsymbol{\alpha}}_t \\ \tilde{\boldsymbol{\alpha}}_t & 1 \end{bmatrix}, \tag{2.2.15}$$

where $\tilde{\boldsymbol{\alpha}}_t = \tilde{K}_{t-1}^{-1}\tilde{\mathbf{k}}_{t-1}(\mathbf{s}_t)$ (2.1.11) and $\delta_t = 1 - \tilde{\mathbf{k}}_{t-1}(\mathbf{s}_t)^\top \tilde{\boldsymbol{\alpha}}_t$ (2.1.12). After performing the above update, we set $\tilde{\boldsymbol{\alpha}}_t$ equal to $(0, \ldots, 0, 1)^\top$. This happens as the dictionary $\mathcal{D}_t$ contains the state $\mathbf{s}_t$ and consequently its feature vector $\boldsymbol{\phi}(\mathbf{s}_t)$ can be exactly represented by itself. Moreover, matrices A_t and H_t are now formed as:

$$A_t = \left[\begin{array}{cc} A_{t-1} & \mathbf{0} \\ \hline \tilde{\boldsymbol{\alpha}}_t \end{array}\right] \quad \text{and} \quad H_t = \left[\begin{array}{cc} H_{t-1} & \mathbf{0} \\ \hline \mathbf{h}_t \end{array}\right], \tag{2.2.16}$$

where $\mathbf{h}_t = (0, \ldots, 0, 1, -\gamma)^\top$ and $\mathbf{0}$ is a vector of zeros of appropriate length. Therefore,

$$\tilde{H}_t = H_t A_t = \begin{bmatrix} \tilde{H}_{t-1} & \mathbf{0} \\ \tilde{\boldsymbol{\alpha}}_{t-1}^\top & -\gamma \end{bmatrix} = \left[\begin{array}{cc} \tilde{H}_{t-1} & \mathbf{0} \\ \hline \tilde{\mathbf{h}}_t^\top \end{array}\right], \tag{2.2.17}$$

where

$$\tilde{\mathbf{h}}_t = \begin{bmatrix} \tilde{\boldsymbol{\alpha}}_{t-1} \\ 0 \end{bmatrix} - \gamma \tilde{\boldsymbol{\alpha}}_t = \begin{bmatrix} \tilde{\boldsymbol{\alpha}}_{t-1} \\ \gamma \end{bmatrix}. \tag{2.2.18}$$

Furthermore, we get the following update rule by multiplying the transpose of Eq. 2.2.17 by itself,

$$\begin{aligned}\tilde{H}_t^\top \tilde{H}_t &= \begin{bmatrix} \tilde{H}_{t-1}^\top & \tilde{\boldsymbol{\alpha}}_{t-1} \\ \mathbf{0}^\top & -\gamma \end{bmatrix}\begin{bmatrix} \tilde{H}_{t-1} & \mathbf{0} \\ \tilde{\boldsymbol{\alpha}}_{t-1}^\top & -\gamma \end{bmatrix} \\ &= \begin{bmatrix} \tilde{H}_{t-1}^\top \tilde{H}_{t-1} + \tilde{\boldsymbol{\alpha}}_{t-1}\tilde{\boldsymbol{\alpha}}_{t-1}^\top & -\gamma\tilde{\boldsymbol{\alpha}}_{t-1} \\ -\gamma\tilde{\boldsymbol{\alpha}}_{t-1}^\top & \gamma^2 \end{bmatrix}. \end{aligned} \tag{2.2.19}$$

Thus, using the partitioned matrix formula (Appendix A.2), we get the following recursive formula for P_t:

$$P_t = \begin{bmatrix} P_{t-1} & \frac{1}{\gamma} P_{t-1}\tilde{\boldsymbol{\alpha}}_{t-1} \\ \frac{1}{\gamma}\tilde{\boldsymbol{\alpha}}_{t-1}^\top P_{t-1} & \frac{1}{\gamma^2}\tilde{\boldsymbol{\alpha}}_{t-1}^\top P_{t-1}\tilde{\boldsymbol{\alpha}}_{t-1} + \frac{1}{\gamma^2} \end{bmatrix}. \tag{2.2.20}$$

Finally, we derive the following recursive update rule for $\mathbf{v}_t$:

$$\begin{aligned} \mathbf{v}_t = \tilde{H}_t^\top R_{t-1} &= \left[\begin{array}{c|c} H_{t-1}^\top & \tilde{\mathbf{h}}_t \\ \mathbf{0}^\top & \end{array}\right] \begin{bmatrix} R_{t-2} \\ r_{t-1} \end{bmatrix} \\ &= \begin{bmatrix} H_{t-1}^\top R_{t-2} \\ 0 \end{bmatrix} + \tilde{\mathbf{h}}_t r_{t-1} \\ &= \begin{bmatrix} \mathbf{v}_{t-1} \\ 0 \end{bmatrix} + \tilde{\mathbf{h}}_t r_{t-1}. \end{aligned} \tag{2.2.21}$$

Therefore, we achieve to convert the reinforcement learning policy evaluation problem to a regression problem, by deriving recursive formulas for the computation of P_t, $\tilde{K}_t$ and $\mathbf{v}_t$ parameters. It is also worth noting that the overall computational cost is bounded by $O(|\mathcal{D}_t|^2)$, per time step. Therefore, assuming that we bound the size of the final dictionary, our scheme meets all the online reinforcement learning requirements. The pseudocode for this scheme is illustrated in Algorithm 2.2. The only remaining question is how to calculate the unknown parameters, $\tilde{\mathbf{w}}_t$, of the linear regression model (2.2.8). In this work, we adopt a Bayesian sparse kernel technique, which leads to sparse solutions whilst maintaining high generalization capabilities.

2.2.2 Sparse Bayesian Regression

In the previous section, we have shown how the policy evaluation problem can be converted into a regression problem through a sequential kernel sparsification scheme. The only open issue now for the completion of the policy evaluation problem is the approximation of the value function of a given policy, π. At this point, we describe the Bayesian optimization scheme adopted for the estimation of the unknown model parameters, $\tilde{\mathbf{w}}_t$. For notation simplicity, the linear regression model of Eq. 2.2.8 can be written more concisely as:

$$\mathbf{y}_t = \tilde{K}_t \tilde{\mathbf{w}}_t + \mathbf{e}_t, \tag{2.2.22}$$

where $\mathbf{y}_t \triangleq P_t \mathbf{v}_t$ represents the observations of our model. The term $\mathbf{e}_t$ plays the role of the stochastic model noise and is assumed to be a zero-mean Gaussian distribution with precision β_t, i.e. $\mathbf{e}_t \sim \mathcal{N}(0, \beta_t^{-1} I)$. In view of this, the conditional probability density of the sequence $\mathbf{y}_n$ is also Gaussian, i.e.

$$p(\mathbf{y}_t | \tilde{\mathbf{w}}_t, \beta_t) = \mathcal{N}(\mathbf{y}_t | \tilde{K}_t \tilde{\mathbf{w}}_t, \beta_t^{-1} I). \tag{2.2.23}$$

An important issue is how to define the optimal order of the above regression model. Sparse Bayesian methodology offers an advanced solution to this problem, by penalizing

Algorithm 2.2: Relevance Vector Machine Temporal Difference Algorithm

Input : $\nu, \gamma, \boldsymbol{s}_1$;
Output : $\mathcal{D}_t, \tilde{K}_t, \mathbf{v}_t, P_t$;
Initialize: $\mathcal{D}_1 = \{\boldsymbol{s}_1\}$, $\tilde{K}_1 = 1$, $\tilde{K}_1^{-1} = 1, P_1 = 1, \mathbf{v}_1 = 0, \tilde{\boldsymbol{\alpha}}_1 = 1$;

for $t = 2, 3, \ldots$ **do**
  Observe transition $\boldsymbol{s}_t$, r_{t-1};
  $\tilde{\boldsymbol{\alpha}}_t = \tilde{K}_{t-1}^{-1} \tilde{\mathbf{k}}_{t-1}(\boldsymbol{s}_t)$;
  $\delta_t = 1 - \tilde{\mathbf{k}}_{t-1}(\boldsymbol{s}_t)^\top \tilde{\boldsymbol{\alpha}}_t$;
  if $\delta_t > \nu$ **then**
    $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{\boldsymbol{s}_t\}$;
    Compute $\tilde{K}_t$ (Eq. 2.2.14);
    Compute $\tilde{K}_t^{-1}$ (Eq. 2.2.15);
    $P_t = \begin{bmatrix} P_{t-1} & \frac{1}{\gamma} P_{t-1} \tilde{\boldsymbol{\alpha}}_{t-1} \\ \frac{1}{\gamma} \tilde{\boldsymbol{\alpha}}_{t-1}^\top P_{t-1} & \frac{1}{\gamma^2} \tilde{\boldsymbol{\alpha}}_{t-1}^\top P_{t-1} \tilde{\boldsymbol{\alpha}}_{t-1} + \frac{1}{\gamma^2} \end{bmatrix}$;
    $\tilde{\mathbf{h}}_t = \begin{bmatrix} \tilde{\boldsymbol{\alpha}}_{t-1} \\ \gamma \end{bmatrix}$;
    $\mathbf{v}_t = \begin{bmatrix} \mathbf{v}_{t-1} \\ 0 \end{bmatrix} + \tilde{\mathbf{h}}_t r_{t-1}$;
    $\tilde{\boldsymbol{\alpha}}_t = (0, \ldots, 0, 1)^\top$;
  else
    $\tilde{\mathbf{h}}_t = \tilde{\boldsymbol{\alpha}}_{t-1} - \gamma \tilde{\boldsymbol{\alpha}}_t$;
    $P_t = P_{t-1} - \frac{P_{t-1} \tilde{\mathbf{h}}_t \tilde{\mathbf{h}}_t^\top P_{t-1}}{1 + \tilde{\mathbf{h}}_t^\top P_{t-1} \tilde{\mathbf{h}}_t}$;
    $\mathbf{v}_t = \mathbf{v}_{t-1} + \tilde{\mathbf{h}}_t r_{t-1}$;
  end
end

large order models. This is the idea behind the Relevance Vector Machines [129]. More specifically, a heavy-tailed prior distribution, $p(\tilde{\mathbf{w}}_t)$, is imposed over the regression coefficients $\tilde{\mathbf{w}}_t$ to zero out most of the weights after the learning process. This is achieved in a two-phase hierarchical way:

- Firstly, a zero-mean Gaussian distribution is considered over vector parameters $\tilde{\mathbf{w}}_t$:

$$p(\tilde{\mathbf{w}}_t | \Xi_t) = \mathcal{N}(\tilde{\mathbf{w}}_t | \mathbf{0}, \Xi_t^{-1}) = \prod_{i=1}^{|\mathcal{D}_t|} \mathcal{N}(\tilde{w}_i | 0, \xi_i^{-1}), \tag{2.2.24}$$

  where Ξ_t is a diagonal matrix of size $|\mathcal{D}_t| \times |\mathcal{D}_t|$, with the elements of the precision vector $\boldsymbol{\xi}_t = (\xi_1, \ldots, \xi_{|\mathcal{D}_t|})^\top$ on the main diagonal. Thus, a separate hyper-parameter is introduced to each weight parameter $\tilde{w}_i$, instead of a single common hyper-parameter.

- Secondly, a Gamma hyperprior is imposed over each hyper-parameter ξ_i:

$$p(\boldsymbol{\xi}_t) = \prod_{i=1}^{|\mathcal{D}_t|} \mathcal{G}amma(\xi_i|a,b). \tag{2.2.25}$$

  It must be noted that both Gamma parameters a and b are *a priori* set to zero in order to make these priors uninformative.

This two-stage hierarchical prior is actually a Student's-t distribution that provides sparseness to the model [129], since it enforces most of the parameters ξ_i to become large and as a result the corresponding weights $\tilde{w}_i$ are set to zero. Therefore, the complexity of the regression model is controlled automatically, while at the same time overfitting is avoided. Furthermore, we obtain the marginal likelihood distribution of sequence $\mathbf{y}_t$, by integrating out the weights $\tilde{\mathbf{w}}_t$, which is also a zero mean Gaussian distribution:

$$p(\mathbf{y}_t|\boldsymbol{\xi}_t, \beta_t) = \int p(\mathbf{y}_t|\tilde{\mathbf{w}}_t, \beta_t) p(\tilde{\mathbf{w}}_t|\boldsymbol{\xi}_t) d\tilde{\mathbf{w}} = \mathcal{N}(\mathbf{0}, C_t), \tag{2.2.26}$$

where the covariance matrix has the form $C_t \triangleq \tilde{K}_t \Xi_t^{-1} \tilde{K}_t^\top + \beta_t^{-1} I$.

From the Bayes rule, the posterior distribution of the weights is again Gaussian and takes the following form [129]:

$$p(\tilde{\mathbf{w}}_t|\mathbf{y}_t, \boldsymbol{\xi}_t, \beta_n) = \mathcal{N}(\tilde{\mathbf{w}}_t|\boldsymbol{\mu}_t, \Sigma_t), \tag{2.2.27}$$

where the mean and covariance are respectively given by:

$$\boldsymbol{\mu}_t = \beta_t \Sigma_t \tilde{K}_t^\top \mathbf{y}_t, \tag{2.2.28}$$

$$\Sigma_t = (\beta_t \tilde{K}_t^\top \tilde{K}_t + \Xi_t)^{-1}. \tag{2.2.29}$$

The maximization of the log-likelihood function given by Eq. 2.2.26 leads to the following update rules for the model parameters [129]:

$$\xi_i = \frac{\gamma_i}{\mu_i^2}, \tag{2.2.30}$$

$$\beta_t^{-1} = \frac{\|\mathbf{y}_t - \tilde{K}_t \boldsymbol{\mu}_t\|^2}{n - \sum_{i=1}^n \gamma_i}, \tag{2.2.31}$$

where $n = |\mathcal{D}_t|$. The quantity γ_i is defined as $\gamma_i \triangleq 1 - \xi_i [\Sigma_t]_{ii}$ and measures how well the corresponding parameter $\tilde{w}_i$ is determined by the data. Moreover, $[\Sigma_t]_{ii}$ denotes the i-th diagonal element of the posterior covariance Σ_t (Eq. 2.2.29). Therefore, the learning algorithm is involved by iterated applications of Equations (2.2.28), (2.2.29), (2.2.30) and (2.2.31) until a suitable convergence criterion is satisfied. Finally, when the learning procedure converges, the value function of the new state s is evaluated based on the predictive distribution which is also Gaussian and is given by:

$$(\hat{V}^\pi(\boldsymbol{s})|\mathbf{y}_t) \backsim \mathcal{N}\left(\hat{V}^\pi(\boldsymbol{s})|\boldsymbol{\mu}_t^\top \tilde{\mathbf{k}}_t(\boldsymbol{s}), \beta_t^{-1} + \tilde{\mathbf{k}}_t(\boldsymbol{s})^\top \Sigma_t \tilde{\mathbf{k}}_t(\boldsymbol{s})\right). \tag{2.2.32}$$

The hyper-parameters are set to their optimized values obtained at the end of the learning procedure.

Incremental Learning

Despite the efficiency of the standard RVM scheme, its application in large scale problems is prohibitive since inversion of matrix Σ_n (Eq. 2.2.29) is required at each iteration step of the learning process. In our approach, this becomes obvious when the dictionary size ($|\mathcal{D}_t|$) becomes quite large. To overcome this handicap, we follow an incremental learning algorithm that has been proposed in [130] and offers a significantly faster procedure for hyper-parameters optimization.

Initially, the specific approach assumes that all dictionary states (all basis functions) have been pruned from the model due to the sparsity constraint. This is equivalent to assuming that $\xi_i = \infty$, $\forall i = \{1, \ldots, |\mathcal{D}_t|\}$. Then, at each iteration, a basis function is examined whether to be added to the model or removed from the model or re-estimated. Adding a basis function to the model, the value of the hyper-parameter ξ_i is estimated according to the maximum likelihood criterion. In particular, it is easily verified that the term of the marginal likelihood of Eq. 2.2.26, which refers to the hyper-parameter ξ_i, is given by:

$$\ell(\xi_i) = \frac{1}{2}\left(\log \xi_i - \log(\xi_i + s_i) + \frac{q_i^2}{\xi_i + s_i}\right), \tag{2.2.33}$$

where

$$s_i = \frac{\xi_i S_i}{\xi_i - S_i}, \qquad q_i = \frac{\alpha_i Q_i}{\alpha_i - S_i}, \tag{2.2.34}$$

and $S_i = \tilde{\mathbf{k}}_i^\top C_t^{-1} \tilde{\mathbf{k}}_i$, $Q_i = \tilde{\mathbf{k}}_i^\top C_t^{-1} \mathbf{y}_t$, $\tilde{\mathbf{k}}_i = (k(\tilde{\mathbf{s}}_1, \tilde{\mathbf{s}}_i), \ldots, k(\tilde{\mathbf{s}}_{|\mathcal{D}_t|}, \tilde{\mathbf{s}}_i))^\top$. It is worth noting that the matrix inversion is avoided by using the matrix inversion lemma (Appendix A.2). As it has been shown in [130], the log-likelihood has a single maximum at:

$$\xi_i = \frac{s_i^2}{q_i^2 - s_i}, \quad \text{if } q_i^2 > s_i, \tag{2.2.35}$$

$$\xi_i = \infty, \qquad \text{if } q_i^2 \leq s_i. \tag{2.2.36}$$

Thus, a basis function $\tilde{k}_i$ is added to the model in the case of $q_i^2 > s_i$ (the corresponding state of the dictionary becomes active). Otherwise, the specific basis function is removed from the model.

2.2.3 Episodic tasks

In the preceding sections, we have presented the online RVMTD algorithm for infinite horizon problems (continuing tasks), where the agent is placed initially at a random state and then is let to wander-off indefinitely. Since most of the RL tasks are episodic, a modification to our approach is required to handle episodic learning tasks. In this case, our approach has to consider a series of episodes, rather than a unique sequence of time steps. During an episodic task, the agent follows a trajectory until it reaches an absorbing state after a finite number of time steps. When an absorbing state is reached, a new episode (epoch) starts and the agent is placed in a usually unknown random initial position.

An absorbing state may be thought as a special state that only zero-rewards are generated and actions do not have any impact since transitions lead to the same state. In this case, once an absorbing state is reached, all subsequent rewards vanish. Therefore, the discounted return of the state preceding the terminal state is equal to the immediate received reward. In this direction, a zero reward is assumed to be received during the transition from terminal state to the starting state of the next episode. Practically, this is achieved by setting the discount factor γ temporarily to zero, every time the agent reaches a terminal state. Therefore, the matrix H_t gets the following form:

$$H_t = \begin{bmatrix} 1 & -\gamma & 0 & \cdots & 0 & 0 \\ 0 & 1 & -\gamma & \cdots & 0 & 0 \\ \vdots & \vdots & & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -\gamma & 0 \\ 0 & 0 & \cdots & 0 & 1 & 0 \end{bmatrix}. \tag{2.2.37}$$

This new form of matrix H_t is the only modification we make to our approach in order to deal with episodic tasks. After a sequence of episodes, matrix H_t becomes a square block diagonal matrix, with each block corresponding to a completed episode.

2.2.4 Relevance Vector Machine TD for Policy Improvement

Until now, our research is mainly focused on the policy evaluation problem. In this section, we turn our attention to the control learning problem, where the agent has to discover a (near) optimal policy, while interacting with an unknown environment. As the MDP's transition model is unknown in advance, we need to learn the action-value function (Q) rather than the state-value function (V), in order to find out an optimal or near-optimal policy.

In light of this, we have considered transitions between state-action pairs instead of transitions between single states, since the model environment is completely unknown. This is achieved by defining a kernel function over state-action pairs, i.e., $k : (\mathcal{S} \times \mathcal{A}) \times (\mathcal{S} \times \mathcal{A}) \to \mathbb{R}$. Since states and actions are independent entities, the kernel function can be considered as a product of a state kernel k_s and an action kernel k_a, i.e., $k(\boldsymbol{s}, a, \boldsymbol{s}', a') = k_s(\boldsymbol{s}, \boldsymbol{s}')k_a(a, a')$ (legitimate kernel, [111, 17]). Using the new state-action kernel function, the RVMTD algorithm approximates the action-value function, $Q^\pi(\boldsymbol{s}, a)$, of the policy used to make decisions, π. In addition, our approach makes use of the ϵ-greedy exploration scheme[3] for selecting an action at each time step. Consequently, actions are selected greedily (choosing the action with the maximum action-value estimation) most of the time, while with a small probability ϵ, actions are selected uniformly random. In this way, RVMTD can be considered an on-policy reinforcement learning algorithm, since the estimated policy is used to make decisions at each time step.

[3]Different exploration schemes may be used, e.g. softmax.

2.3 Empirical Evaluation

We have conducted two sets of experiments to analyse the online performance of the RVMTD algorithm. Comparisons have been made with an online variant of GPTD [46] for policy improvement over 1000 episodes. Both algorithms are based on the same online kernel sparsification methodology, where the threshold parameter ν controls the level of sparsity of the solution. In all experiments, we set the sparsity factor ν equal to 0.1.

A covariance kernel function is also defined over state-action pairs (see Section 2.2.4). A Gaussian kernel is used as the kernel function over the state space $\mathcal{S}$, with a different scalar parameter for each dimension. Furthermore, a simple kernel has also been used over the action space $\mathcal{A}$, which is equal to 1 in the case of similar actions and 0 otherwise[4]. Finally, actions are selected ϵ-greedily by choosing the best action, and slowly decreasing ϵ toward zero.

2.3.1 Domains

We have consider two well-known continuous state, discrete-action, episodic domains. The first one is the *mountain car* domain and the second one is the *inverted pendulum* domain.

Mountain Car

The objective in this domain is to drive an under-powered car up a steep mountain road to the top of a hill. Due to the force of gravity, the car cannot accelerate up to the tophill and thus, it must go to the opposite slope to acquire enough momentum, so as to reach the goal on the right slope (at position > 0.5). The state space consists of two continuous variables: the position ($p \in [-1.2, +0.5]$) and the current velocity ($u \in [-0.07, 0.07]$) of the car. There are three actions: reverse, zero, and forward throttle. All three actions are noisy as a uniform noise in $[-0.2, 0.2]$ is added to the chosen action. At each time step, it receives a negative reward $r = -1$ except for the case where the target is reached (zero reward). An episode terminates either when the car reaches the goal at the right tophill, or when the total number of steps exceeds a maximum allowed value (1000). At the beginning of each episode, the vehicle is positioned at a new state, with the position and velocity uniformly randomly selected. The discount factor γ is set to 0.99. Moreover, the algorithms under comparison use a Gaussian state kernel with different scalar parameters. In the case of RVMTD, the scalar parameters of the two variables (i.e., position and velocity), are set to 0.08 and 0.0008, respectively. On the other side, for the GPTD we set the position and velocity scalar parameters equal to 0.04 and 0.0004, respectively. It is worth noting that the GPTD algorithm is quite

[4]In the case of GPTD algorithm in the mountain car domain, the action kernel is equal to 0.6 if actions are adjacent (e.g. forward action is adjacent to neutral action), 1 in the case of similar actions and 0 otherwise.

sensitive to the specific parameters. Last but not least, the prior variance noise of the GPTD algorithm is equal to 1.

Inverted Pendulum

The target in this task is to balance a pendulum by applying forces of a fixed magnitude (50 Newtons). The state space consists of two continuous variables, the vertical angle (θ) and the angular velocity ($\dot{\theta}$) of the pendulum. At each time step, the agent has at his arsenal three actions: no force, left force and right force. All three actions are noisy as a uniform noise in $[-10, 10]$ is added to the chosen action. A zero reward is received at each time step except for the case where the pendulum falls. In this case, a negative reward ($r = -1$) is given and a new episode begins. An episode also ends with a reward of $r = 0$ after 3000 steps. In this case, we consider that the pendulum is successfully balanced. Each episode starts by setting the pendulum in a perturbed state close to the equilibrium point. More information about the specific dynamics can be found in [75]. The discount factor γ is set to 0.95. In the case of the Gaussian kernel state, we set the scalar parameters for the angle and velocity of the pendulum equal to 0.01 and 0.25, respectively. Finally, the GPTD parameter for the prior variance noise is set equal to 0.1.

2.3.2 Results

In our results, we show the average performance in terms of the number of steps of each method, averaged over 100 runs. For each average, we also plot the 95% confidence interval for the accuracy of the mean estimate with error bars. In addition, we present the 90% percentile region of the runs, in order to indicate the inter-run variability in performance.

Figure 2.1 shows the results of the experiments in both *mountain car* and *pendulum domains*. For the *mountain car*, it is clear that RVMTD significantly outperforms GPTD algorithm. Another interesting remark is the ability of RVMTD to discover a good or near-optimal policy in a much higher rate compared to the GPTD. Furthermore, the RVMTD algorithm presents a more robust behaviour, as it achieves to discover optimal policies at each run. Last but not least, RVMTD generates a much smaller dictionary ($|\mathcal{D}| \simeq 129$) in contrast to that generated by GPTD ($|\mathcal{D}| \simeq 220$).

For the *pendulum*, the performance difference between the two algorithms is even more impressive. More specifically, RVMTD is more stable, while it achieves to discover the optimal policy in all runs. This remark stems from the fact that after 70 episodes more than 90% of the runs are optimal, while many GPTD runs fail to find a good solution even after hundreds of episodes. It is also worth noting that both approaches generate dictionaries of similar size ($|\mathcal{D}| \simeq 230$).

Figure 2.2 illustrates the value functions of the policies learned by the RVMTD algorithm in both *mountain car* and *pendulum*, at the end of a single successful run

(i.e., 1000 episodes).

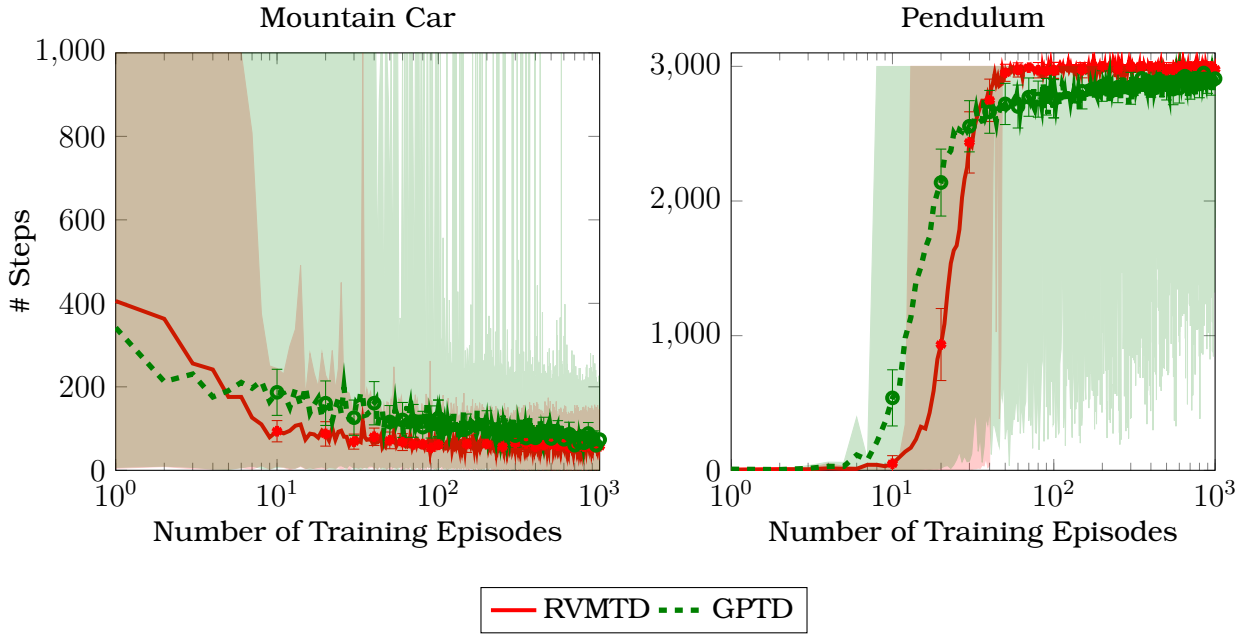


Figure 2.1: Experimental evaluation. The solid line shows RVMTD, while the dashed line shows GPTD. The error bars denote 95% confidence intervals for the mean (i.e., statistical significance). The shaded regions denote 90% percentile performance (i.e., robustness) across runs.

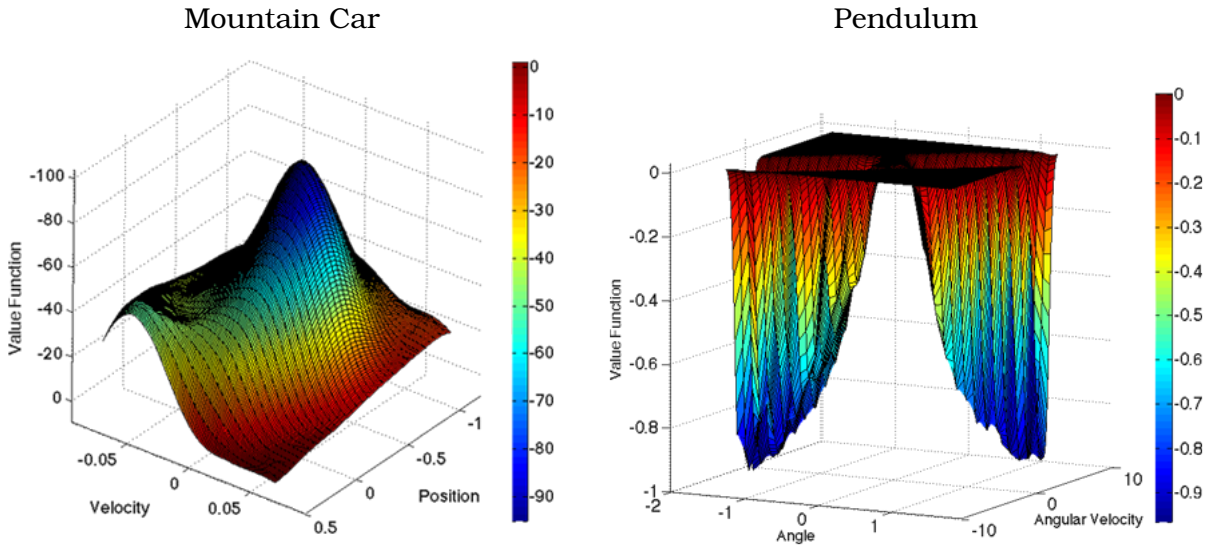


Figure 2.2: Value functions of the learned policies at the end of a single run.

2.4 Summary

In this chapter, we have presented an advanced kernelized Bayesian methodology for model-free value function approximation using the RVM regression framework as the generative model. The key aspect of the proposed technique is the translation of the policy evaluation problem to a regression problem. An online sparsification kernel technique has been adopted in order to render the proposed algorithm practical in large scale domains. Furthermore, we have also proved that recursive formulas can be derived for the update of the model observations at each time step. A sparse Bayesian regression framework has been used for the estimation of the unknown model coefficients, incorporating powerful modeling capabilities. We have also applied an incremental learning strategy that accelerates the optimization procedure. Finally, we have proposed an extension of the RVMTD algorithm for the estimation of action-value functions, allowing the gradual improvement of policies, without knowledge of the model of the environment. As experimental analysis has shown, the proposed RVMTD algorithm is able to achieve better performance than GPTD algorithm.

Chapter 3

Model-based Reinforcement Learning using an Online Clustering Scheme

One of the most important challenges for modeling sequential decision making in stochastic environments is that of finding a suitable representation of the state space. Discovering a good set of features is of central interest for the development of an effective value function approximation scheme, as the quality of the approximation depends on the set of features [133]. This happens due to the fact that the value function is represented as a linear combination of a set of features (basis functions). In this way, states are mapped into feature vectors allowing the approximation of the value function in continuous state spaces. Practically, the selected set of features affects not only the quality of the obtained solution, but also the convergence rate of the learning process.

Generally, the basis functions used for approximating the value function remain fixed during the learning process (see Section 1.2.2). For instance, in the work presented in [73], a number of fixed Fourier basis functions are used for value function approximation. Nevertheless, various approaches have been proposed for the estimation and adaptation of the basis functions during the learning procedure. The specific approaches are able to tackle the problem of finding an appropriate set of features in an automatic way, and can be divided into two distinct categories. The first category relies on the dynamics of the environment (i.e. transition model), which are learned through the interaction of the agent with the environment. More specifically, the observed

transitions between states are used for the construction of a state connectivity graph. Thereafter, spectral clustering methodologies [80] are used for the construction of the state features. The resulting features capture interesting transition properties of the environment (e.g. different spatial resolution) and are reward-independent. The latter property allows the effective re-computation of value functions in the case of problem changes in terms of rewards. On the other hand, the methods that lie in the second category are based on the minimization of the Bellman error for the construction of an appropriate set of basis functions [69, 95]. In this case, features are reward-oriented, and are formed based on the reduction of the value function estimation error. In the work presented in [85], a set of k RBF basis function are adjusted directly over the Bellman's equation of the value function. Furthermore, in [99] the probability density function and the reward model, which are assumed to be known, are used for creating basis function from Krylov space vectors (powers of the transition matrix used to systems of linear equations). However, in contrast to the features constructed by the methods of the former category, in this case the resulting features cannot be easily interpreted.

In this chapter, we propose a novel model-based framework for value function approximation which addresses the online construction of a number of appropriate basis functions. An online version of the Expectation-Maximization (EM) algorithm is used for separating the input space (i.e. state-action pairs) into clusters. Furthermore, a mechanism for automatically adding clusters has also been proposed. In this way, a number of clusters are automatically constructed and updated through the interaction of the agent with the environment. Based on the structure of the created clusters, we develop a dictionary of basis functions which are subsequently used for the approximation of the value function. At the same time, an approximate model of the environment is learnt, by keeping useful statistics for each cluster. Thus, we learn the transition probabilities between clusters as well as their expected returns. Policy is then evaluated at each time step by estimating the linear weights of the value function through the least-squares solution. The above learning procedure is repeated after each transition to a new state, improving the policy followed by the agent. The proposed method has been tested on several simulated and real environments where comparisons have been made with an online version of LSPI algorithm that uses a fixed number of basis functions.

The remainder of this chapter is organised as follows. In Section 3.1, we initially introduce the concept of cluster analysis. Sections 3.1.1 and 3.1.2 present the proposed online clustering scheme for features selection, which are used for value approximation. The proposed model-based reinforcement learning algorithm for policy evaluation is presented in Section 3.2. The empirical analysis on simulated and real environments is discussed in Section 3.3. Finally, Section 3.4 summarizes the chapter.

3.1 Clustering for Feature Selection

Clustering (or *cluster analysis*) aims to organize a collection of data items into clusters, such that items within a cluster are more *similar* to each other than they are to items in the other clusters. This notion of similarity can be expressed in very different ways, according to the purpose of the study, to domain-specific assumptions and to prior knowledge of the problem. Clustering is usually performed when no information is available concerning the membership of data items to predefined classes. For this reason, clustering is traditionally seen as part of *unsupervised learning* and can be viewed as the learning of a hidden data concept. Some application areas related with clustering are computer vision, pattern recognition, information retrieval, data mining, bioinformatics, etc. Several taxonomies of clustering methods have been suggested within the literature [43, 17]. A rough but widely agreed frame is to classify clustering techniques as *hierarchical* and *partitional* clustering, based on the properties of clusters generated.

Hierarchical clustering aims to obtain a hierarchy of clusters, called *dendrogram*, that indicates how the clusters are related to each other. These methods proceed either by iteratively merging small clusters into larger ones (agglomerative algorithms) or by splitting large clusters (divisive algorithms). A partition of the data items can be obtained by cutting the dendrogram at a desired level. Agglomerative algorithms need criteria for merging small clusters into larger ones. Most of the criteria concern the merging of pairs of clusters (thus producing binary trees) and are variants of the classical single-link, complete-link or minimum-variance criteria [43].

Partitional clustering aims to directly obtain a single partition of the collection of items into clusters. Many of these methods are based on the iterative optimization of a criterion function reflecting the 'agreement' between the data and the partition. Some important categories of partitional clustering methods are:

- *Methods using the squared error* rely on the possibility to represent each cluster by a prototype and attempt to minimize a cost function that is the sum over all the data items of the squared distance between the item and the prototype of the cluster it is assigned to. In general, the prototypes are the cluster centroids, as in the popular k-means algorithm and its extension k-medoid, fuzzy c-means etc.

- *Density-based methods* consider that clusters are dense sets of data items separated by less dense regions; clusters may have arbitrary shape and data items can be arbitrarily distributed. Such methods rely on the study of the density of items in the neighborhood of each item. Many of the graph-theoretic clustering methods are also related to density-based clustering. The data items are represented as nodes in a graph and the dissimilarity between two items is the 'length' of the edge between the corresponding nodes. Spectral clustering is a kind of such techniques based on similarity information between data points. That is, similar data points (or points with high affinity) are more likely to belong to the

same cluster than points with low affinity.

- *Model-based clustering methods*, assume that the data are generated from a mixture of probability distributions with each component corresponding to a different cluster [82, 17]. In these methods the expectation-maximization (EM) algorithm [36] is the most frequent choice for solving the estimation problem and tuning the mixture parameters.

A fundamental issue in clustering is the determination of the number of clusters in a given data set. Most clustering methods assume that this number is a priori known, and then try to place K clusters in the space defined by the data. In the literature quite a few methods have been proposed for determining the number of clusters. A common approach is to apply a clustering technique using a range of values for K and select the solution that performs best according to certain evaluation (information-based) criteria, such as the Akaike information criterion, the Bayesian information criterion and the cross-validation criterion. Under the Bayesian framework model selection is treated by the posterior distribution for the model uncertainty. Finally, other numerical approximation schemes have been considered, for calculating the posterior distribution, such as the Laplace approximation, Markov chain Monte Carlo (MCMC) methods, and variational approaches (for a brief survey, see [52, 60]).

3.1.1 Clustering using Mixture Models

In this section we present the proposed model-based reinforcement learning scheme which is based on a policy evaluation scheme that incrementally separates the input space into clusters and estimates the transition probabilities among them. Thus, a dictionary of features is dynamically constructed for modeling the value functions. In the following, we assume that the input samples are state-action pairs, denoted as $\boldsymbol{x}_n = (\boldsymbol{s}_n, a_n)$. We also consider a finite action space of size M.

Let us suppose that a data set of $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ is given, consisting of N samples. The task of clustering aims at partitioning the input set into k disjoint clusters, with each cluster to disclose the intrinsic structures of the data. Mixture modeling [17] offers a convenient and at the same time elegant framework for clustering, where the properties of a single cluster c_j, are implicitly described via a probability distribution with parameters θ_j. Therefore, mixture modeling can be formulated as a linear superposition of k probability distributions given by:

$$p(\boldsymbol{x}|\Theta_k) = \sum_{j=1}^{k} u_j p(\boldsymbol{x}|\theta_j), \tag{3.1.1}$$

where Θ_k denotes the set of mixture model parameters. Each one of the probability densities is called as a *component* of the mixture model. The parameters u_j, for $j =$

$1, \ldots, k$, represent the mixing coefficients and must satisfy the following property:

$$\sum_{j=1}^{k} u_j = 1, \tag{3.1.2}$$

together with $0 \leq u_j \leq 1$, in order to yield valid probabilities. Since two uncorrelated sources of information are incorporated in the data (state-action) of our scheme, we consider that the conditional density for each one of the clusters can be written as the product of two probability density functions:

- A Gaussian pdf for the state space:

$$\mathcal{N}(\boldsymbol{s}|\boldsymbol{\mu}_j, \Sigma_j) \propto \exp\left\{-\frac{1}{2}(\boldsymbol{s}-\boldsymbol{\mu}_j)^\top \Sigma_j^{-1}(\boldsymbol{s}-\boldsymbol{\mu}_j)\right\}. \tag{3.1.3}$$

- A Multinomial pdf for the action space:

$$\textit{Multinomial}\,(a|\boldsymbol{\rho}_j) = \prod_{i=1}^{M} \rho_{ji}^{I(a,i)}, \tag{3.1.4}$$

  where $\boldsymbol{\rho}_j$ is an $M \times 1$ probabilistic vector ($\sum_{i=1}^{M} \rho_{ji} = 1$), and $I(a,i)$ is a binary indicator function, defined as:

$$I(a,i) = \begin{cases} 1, & \text{if } \ a = i \\ 0, & \text{otherwise} \end{cases}.$$

Thus, the probability of $\boldsymbol{x}$ conditioned on cluster c_j is given by:

$$p(\boldsymbol{x}|\theta_j) = \mathcal{N}(\boldsymbol{s}|\boldsymbol{\mu}_j, \Sigma_j)\; \textit{Multinomial}\,(a|\boldsymbol{\rho}_j)\;, \tag{3.1.5}$$

where $\theta_j = \{\boldsymbol{\mu}_j, \Sigma_j, \boldsymbol{\rho}_j\}$ denotes the set of the model component parameters.

Mixture modeling treats clustering as an estimation problem for the model parameters, $\Theta_k = \{u_j, \theta_j\}_{j=1}^{k}$. This is achieved by maximizing the log-likelihood function given as:

$$\ell(\Theta_k) = \sum_{n=1}^{N} \log\left\{\sum_{j=1}^{k} u_j\, \mathcal{N}(\boldsymbol{s}_n|\boldsymbol{\mu}_j, \Sigma_j)\; \textit{Multinomial}\,(a_n|\boldsymbol{\rho}_j)\right\}\;. \tag{3.1.6}$$

The Expectation-Maximization (EM) algorithm [36] is an efficient framework that can be used for the estimation of the model parameters. Its main characteristic is that it leads to closed-form update rules for the model parameters (see [17] for an overview). More specifically, the EM algorithm iteratively alternates between an expectation step and a maximization step, as described below:

- *E-step*: Evaluate the current posterior probabilities of a sample belonging to a cluster, by using the current values of the parameters:

$$z_{nj} = \frac{u_j p(x_n|\theta_j)}{\sum_{j'=1}^{k} u_{j'} p(x_n|\theta_{j'})}. \tag{3.1.7}$$

- *M-step*: Re-estimate model parameters by maximizing the expected complete log-likelihood.

3.1.2 Online EM Learning

Since in reinforcement learning the samples are non-stationary and arrive sequentially, the standard EM algorithm cannot be used. In this section, we present an extension of the standard EM scheme [89] for online estimating mixture models that meets our needs. The proposed online clustering scheme consists of two main phases. Firstly, we decide whether a new cluster must be created or not, by using a decision mechanism. Secondly, the main EM procedure is performed for adjusting the cluster parameters, so as to incorporate a received sample.

Let's assume that a random sample $\boldsymbol{x}_n = (\boldsymbol{s}_n, a_n)$ has been observed and that k clusters have been created until that point. Initially, the proposed scheme performs the *expectation* step (E-step) and calculates the posterior probability values z_{nj} (Eq. 3.1.7), according to the current k-order mixture model. The winner cluster $c_{j^*} \in \{c_1, \ldots, c_k\}$ is then selected according to the maximum posterior value, i.e.,

$$c_{j^*} = \arg\max_{j=1}^{k}\{z_{nj}\}. \tag{3.1.8}$$

We assume here that the degree of belongingness of sample $\boldsymbol{x}_n$ in the cluster c_{j^*} is given by a kernel function $k(\boldsymbol{x}_n, c_{j^*})$. As mentioned previously, the input sample incorporates information of two different sources (e.g., $\boldsymbol{x}_n = (\boldsymbol{s}_n, a_n)$). In this case, the specific kernel function can be written as a product of two kernel functions, one for each type of the input (state-action):

$$k(\boldsymbol{x}_n, c_{j^*}) = k_s(\boldsymbol{x}_n, c_{j^*})k_a(\boldsymbol{x}_n, c_{j^*}) \ , \tag{3.1.9}$$

where

$$k_s(\boldsymbol{x}_n, c_{j^*}) = \exp\{-0.5(\boldsymbol{s}_n - \boldsymbol{\mu}_{j^*})^\top \Sigma_{j^*}^{-1}(\boldsymbol{s}_n - \boldsymbol{\mu}_{j^*})\}$$

is the kernel of the state space, and

$$k_a(\boldsymbol{x}_n, c_{j^*}) = \prod_{i=1}^{M} \rho_{j^*i}^{I(a_n,i)}$$

is the kernel of the action space. If the degree of belongingness of sample $\boldsymbol{x}_n$ in cluster c_{j^*} is less than a predefined threshold value ν, a new cluster $(k+1)$ is created. Then, the created cluster is properly initialized as follows:

$$\begin{aligned}
\boldsymbol{\mu}_{k+1} &= \boldsymbol{s}_n, \\
\Sigma_{k+1} &= \frac{1}{2}\Sigma_{j^*}, \\
\rho_{k+1,i} &= \begin{cases} \xi, & \text{if } \ i = a \\ \frac{1-\xi}{M-1}, & \text{otherwise} \end{cases},
\end{aligned}$$

where ξ is set to a large value (e.g. $\xi = 0.9$). The *maximization* step (M-step) is then applied, providing a step-wise procedure for the adaptation of the model parameters.

In this case, the following update rules are used:

$$u_j = (1-\lambda)u_j + \lambda z_{nj}, \tag{3.1.10}$$

$$\boldsymbol{\mu}_j = \boldsymbol{\mu}_j + \lambda z_{nj}\boldsymbol{s}_n, \tag{3.1.11}$$

$$\Sigma_j = \Sigma_j + \lambda z_{nj}(\boldsymbol{s}_i - \boldsymbol{\mu}_j)(\boldsymbol{s}_i - \boldsymbol{\mu}_j)^\top, \tag{3.1.12}$$

$$\rho_{ji} = \frac{n_{ji}}{\sum_{l=1}^{M} n_{jl}}, \tag{3.1.13}$$

with $n_{ji} \triangleq n_{ji} + I(a_n, i)z_{nj}$ representing the number of times the action a_n has been selected in cluster c_j. The term λ is a small positive step-size parameter (e.g., $\lambda = 0.09$), known as *learning rate*, which determines to what extent the newly acquired information overrides existing information. Moreover, in stochastic problems (like in our case) the learning rate can decrease over time by a small fraction, so as convergence is achieved. At this point, it is worth noting that in the *maximization* step (M-step) of the standard EM algorithm, the update rules for both Gaussian parameters have the quantity $\sum_{n=1}^{N} z_{nj}$ in their denominator e.g., $\boldsymbol{\mu}_j = \frac{\sum_n z_{nj}\boldsymbol{s}_n}{\sum_n z_{nj}}$. In the online EM algorithm, each new sample contributes to the computation of these parameters by a factor equal to $z_{nj}/\sum_{n=1}^{N} z_{nj}$. Therefore, when the number of observations becomes extremely large, the incoming new sample barely influences the parameters of the mixture model. To overcome this handicap, the above presented update rules are selected, where the contribution of each new samples is determined by the learning rate, λ.

3.2 Model-based Value Function Approximation

As becomes obvious from the discussion in the previous section, the EM-based online clustering approach performs a partitioning of the input (state-action) space into clusters that also have the same *transition dynamics*. Clusters can be supposed as nodes of a directed (not full) graph that communicate with each other. The learning scheme described so far constructs new nodes in this graph, by performing a splitting process. In this case, useful information, such as the frequency of node visits and the distance between adjacent nodes are provided. In another point of view, the proposed scheme can be seen as a type of relocatable action model (RAM), that has been recently proposed in [77] in order to provide a decomposition or factorization of the transition function.

In this section, we present a model value function approximation algorithm for the reinforcement learning control problem that estimates the action-values of each *graph* node (cluster). In this case, a number of useful statistics about the environment model must be learnt and kept in memory. To make our analysis more clear, we assume that a trajectory of transitions $(\boldsymbol{s}_i, a_i, r_i, \boldsymbol{s}_{i+1})$ is observed. During the online clustering process, the following statistics are maintained for each one of the clusters:

- $\bar{t}_{j,j'}$: mean number of time-steps between two successively observed clusters c_j, c_j'.

- $\overline{R}(c_j, c'_j)$: mean return received during the transition from cluster c_j to cluster c'_j.
- $n_{j,j'}$: total number of transitions from cluster c_j to cluster c'_j.
- n_j: total number of visits to cluster c_j.

Based on the above statistics, the following quantities can be easily calculated and used during the process evaluation process:

- The statistic $P(c'_j|c_j)$ represents the transition probability between two (adjacent) clusters. It is calculated based on the relative frequencies, according to the following rule:
$$P(c'_j|c_j) = \frac{n_{j,j'}}{n_j} \tag{3.2.1}$$
- The quantity $\overline{R}(c_j)$ declares the expected return that is received by the agent, starting from cluster c_j and following policy π, until the end of the episode:
$$\overline{R}(c_j) = \sum_{j'} P(c'_j|c_j)\overline{R}(c_j, c'_j). \tag{3.2.2}$$

Based on the above statistics, the *Bellman* equation [102] for the action-value function of cluster c_j, may be written as:

$$Q(c_j) = \overline{R}(c_j) + \sum_{j'}^{k} P(c'_j|c_j)\gamma^{\bar{t}_{j,j'}} Q(c'_j) \ , \tag{3.2.3}$$

where the summation may be only performed over the neighbourhood of node c_j (adjacent nodes c'_j where $P(c'_j|c_j) > 0$). Therefore, a set of k equations are available for the expected returns $\overline{R}(c_j)$ (observations) of the k graph nodes, which can be written in a more comprehensive form as:

$$\overline{\boldsymbol{R}}_k = H_k \mathbf{q}_k \ , \tag{3.2.4}$$

where we have considered that the vector of the action-value functions is expressed by the functional form of a linear model:

$$\mathbf{q}_k = \Phi_k^\top \mathbf{w}_k. \tag{3.2.5}$$

In the proposed algorithmic scheme, the kernel 'design' matrix, $\Phi_k = [\boldsymbol{\phi}(c_1), \ldots, \boldsymbol{\phi}(c_k)]$, is explicitly derived from the online clustering solution. This is achieved by using the kernel function given by Eq. 3.1.9. This results to:

$$[\Phi_k]_{jj'} = \exp\left\{-\frac{1}{2}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_{j'})^\top \Sigma_{j'}^{-1}(\boldsymbol{\mu}_j - \boldsymbol{\mu}_{j'})\right\} \boldsymbol{\rho}_j^\top \boldsymbol{\rho}_{j'} \tag{3.2.6}$$

Furthermore, H_k is a square matrix of size $k \times k$ that keeps the statistics of Eq. 3.2.3. More specifically, at each line j, if two clusters are adjacent ($P(j'|j) > 0$), we get $[H_k]_{jj} = 1$ and $[H_k]_{j,j'} = -P(c'_j|c_j)\gamma^{\bar{t}_{j,j'}}$. Finally, $\overline{\boldsymbol{R}}_{\boldsymbol{k}}$ is a $k \times 1$ vector that represents the

Algorithm 3.1: Online EM Temporal Difference Algorithm

Initialize: $\boldsymbol{x}_1 = (\boldsymbol{s}_1, a_1)$, $k = 1$.

for $t = 2, 3, \ldots,$ **do**
- Execute action a_{t-1};
- Observe transition $\boldsymbol{s}_t, r_{t-1}$;
- $a_t = \arg\max_{i=1}^{M} Q(\boldsymbol{s}_t, a_i)$; `// ϵ-greedy selection`
- $c_{j^*} = \arg\max_{j=1}^{k} \{z_{nj}\}$; `// Find winning cluster`
- **if** $K(\boldsymbol{x}_{t+1}, m_{j^*}) < \nu$ **then**
  - Create new cluster ($k = k + 1$), where $m_k = \boldsymbol{x}_t$;
  - Create new weight w_k with random initialization ($\mathbf{w}_t = \mathbf{w}_t \cup \{w_k\}$);
- **else**
  - Update the prototype m_{j^*} (Eqs. 3.1.10-3.1.13);
- **end**
- Update environment model statistics;
- Update model coefficients $\mathbf{w}_t$ (Eq. 3.2.7);

end

expected returns of each node, i.e., $\overline{\boldsymbol{R}}_{\boldsymbol{k}} = [\overline{R}(1), \ldots, \overline{R}(k)]$. The only remaining issue is the estimation of the linear model unknown coefficients, $\mathbf{w}_k$. This is achieved by applying the least-squares solution, given by:

$$\mathbf{w}_k = (\Phi_k H_k \Phi_k^\top)^{-1} \Phi_k \overline{\boldsymbol{R}}_{\boldsymbol{k}}. \tag{3.2.7}$$

Thus, the action-value function of an input state-action pair $\boldsymbol{x} \triangleq (\boldsymbol{s}, a)$ is estimated as follows:

$$Q(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^\top \mathbf{w}_k, \tag{3.2.8}$$

where $\boldsymbol{\phi}(\boldsymbol{x}) = [k(\boldsymbol{x}, c_1), \ldots, k(\boldsymbol{x}, c_k)]^\top$. The above procedure is repeated each time a transition is performed. The proposed algorithm starts with a single cluster $(k = 1)$ initialized with the first sample observed by the agent. At every time step, the online EM clustering procedure and the policy evaluation stage are sequentially performed. The overall concept of the proposed methodology is described in Algorithm 3.1.

3.3 Empirical Evaluation

A series of experiments have been conducted in a variety of simulated and real environments, in order to study the performance of the proposed model-based approach. Comparisons have been made using the Least Square Temporal Difference (LSTD) [20] algorithm in the case of policy evaluation problem (Boyan's Chain), i.e., the evaluation of value function of a given policy π, and the online Least Square Policy Iteration (LSPI) [27] algorithm in the case of control problem, i.e., discovery of the optimal policy. Both methods are considered state-of-the-art RL algorithms for the policy evaluation and the

control problem, respectively, where the action value function is represented with the functional form of a linear model. In our experimental analysis, an equidistant fixed grid of radial basis functions (RBFs) over the state space has been used for the value function representation, following the suggestions presented in [27]. It is worth noting that in our experiments the state space is small, either 1D or 2D, and the agent has at his arsenal a small discrete set of actions. This fact makes both the LSTD and the LSPI methodologies practically applicable (not a huge number of basis functions) and since they cover all possible state-action pairs, they are capable of finding an optimal solution. The objective of our study is to analyse the ability of the proposed scheme to achieve high performance by using as few basis functions as possible, constructed and adapted based on the online EM clustering procedure as previously described.

In all examined environments, the discount factor γ has been set to 0.999, except for the case of Boyan's chain where $\gamma = 1.0$. Furthermore, threshold ν, which determines whether a new cluster must be created or not, is set to 0.7. Finally, in order to introduce stochasticity into the transitions and at the same time to attain better exploration of the model environment, actions are chosen by using the ϵ-greedy exploration scheme. In our experiments, ϵ is initially set to 0.1 and decreases steadily over time.

3.3.1 Domains

Simulated Environments

The first set of experiments have been made using two well-known continuous state, discrete-action, episodic simulated benchmarks. The first one is the Boyan's chain [20] and the second one is the Puddle world [120] environment.

Boyan's Chain

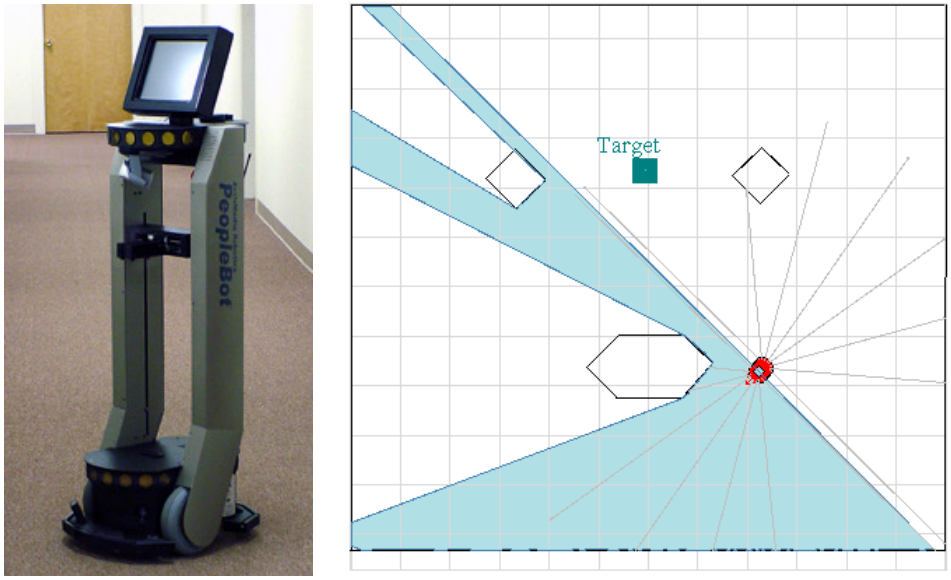
Two chains with a different number of states, $N = 13$ and $N = 98$, respectively, have been examined in our empirical analysis. In the specific domain the states are connected to a chain, where an agent located at a state $s > 2$ can move to states $s - 1$ and $s-2$. After each transition, a reward, $r = -3$, is received by the agent. On the other hand, there are only deterministic transitions from states 2 and 1, to states 1 and 0, respectively, where the received rewards are -2 and 0, respectively. At the beginning of each episode, the agent is located at state N and his objective is to reach state 0 as soon as possible. In Boyan's chain domain, both the policy evaluation problem as well as the problem of discovering the optimal policy (control problem) have been considered. The policy evaluated at the former problem assumes that if an agent is located at a state $s > 2$ he can move to states $s - 1$ or $s - 2$, with the same probability. It must be noted that both LSTD and online LSPI methods have a number of RBF kernels in the state space equal to the number of states with a kernel width equal to 1. In the same way, our algorithm is allowed to construct the same number of clusters as LSTD and LSPI, in an attempt to focus our study on the effect of the cluster transition probabilities.

Puddle World

The Puddle World is a continuous 2-dimensional square world of width 1, found on the RL-Glue Library[1]. In this domain, the objective is to reach the terminal state at the upper right corner starting from a uniformly random position. At the same time, two oval puddles located at the center of the environment must be avoided. More specifically, the puddles consist of the area of radius 0.1 around two line segments, respectively, one from $(0.1, 0.75)$ to $(0.45, 0.75)$, and the other from $(0.45, 0.4)$ to $(0.45, 0.8)$. The agent has at his disposal four actions that correspond to the four cardinal directions: up, down, right and left. Upon selecting a specific action the agent moves 0.05 in the specified direction with an additive Gaussian noise. The received reward is $r = -1$ except for the case, where the agent inserts in a puddle region. In this case, a penalty between 0 and -40 is received, depending on the proximity to the middle of the puddle.

In the particular problem, comparisons have been made with the online LSPI method where an equidistant fixed grid ($N \times N$) of RBFs has been selected for each one of the four actions. In our experiments, three different values of N, $N = \{15, 20, 25\}$, have been used that correspond to 900, 1600, 2500 number of basis functions ($N \times N \times 4$), respectively.

Real Environments



(a) Pioneer/PeopleBot (b) Stage world

Figure 3.1: The mobile robot and the 2D-grid map used in our experiments. This is one snapshot from the MobileSim simulator with visualization of the robot's laser and sonar range scanners.

[1]Available at http://library.rl-community.org/wiki

Experiments have also been conducted in a real world environment, by using the wheeled real mobile robot platform Pioneer/PeopleBot, shown in Fig. 3.1(a), which is based on the robust P3-DX base. The robot is equipped with the Advanced Robot Interface for Applications (ARIA) library that provides a nice framework for controlling and receiving data from the robot platform. At the same time, the robot has at its disposal a number of sensors, such as sonar, laser, bumpers, pan-tilt-zoom camera, etc. For the purposes of our experiments, only the sonar and the laser sensors have been used for obstacle avoidance. Furthermore, an embedded motion controller is able to provide the robot's current state at each time step, such as its position (x, y, θ), range sensing data, etc. Due to numerous physical restrictions (e.g., battery life), the training of the specific methodology is performed by using the MobileSim[2] simulator. MobileSim is built upon the famous Stage platform and manages to simulate the real world environments with satisfactory precision and realism.

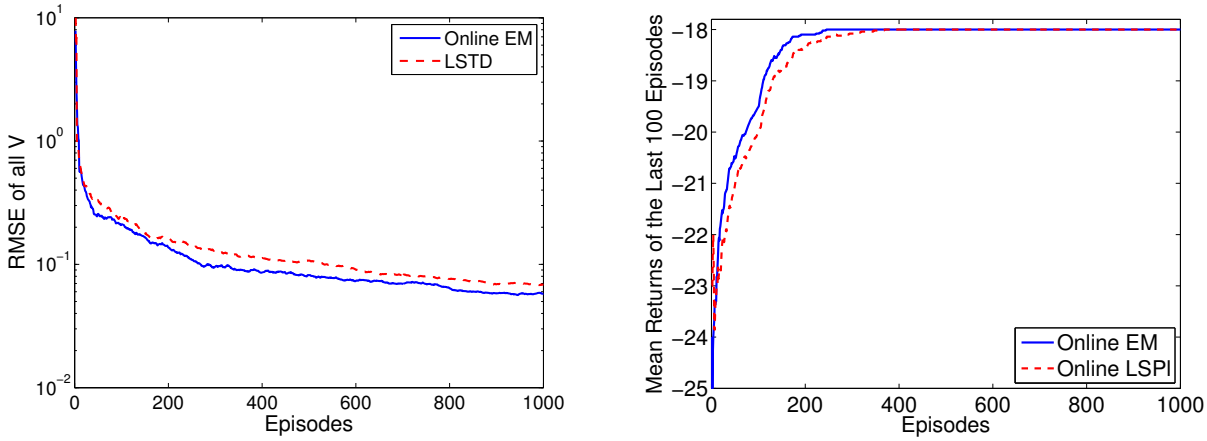
A 2-dimensional grid map (stage world) has been selected for our experiments, as shown in Fig.3.1(b). The specific world has been designed and edited using the Mapper toolkit. In this task, the objective of the robot is to reach a steady landmark (shown with a rectangular green box in the map, Fig.3.1(b)) executing as few steps as possible. At the beginning of each episode, the agent starts from a uniformly random position inside the world and performs a finite number of actions. The particular task is episodic and a new episode begins when one of the following incidents occurs: a maximum number of steps is executed per episode (in our case it is equal to 100), robot hit an obstacle, or the target is reached. The action space has been discretized into 8 major compass winds, while the length of each step is equal to 1 meter. Finally, at each time step the robot receives an immediate reward of $r = -1$, except for the case that an obstacle is hit, where the reward is equal to $r = -100$.

3.3.2 Results

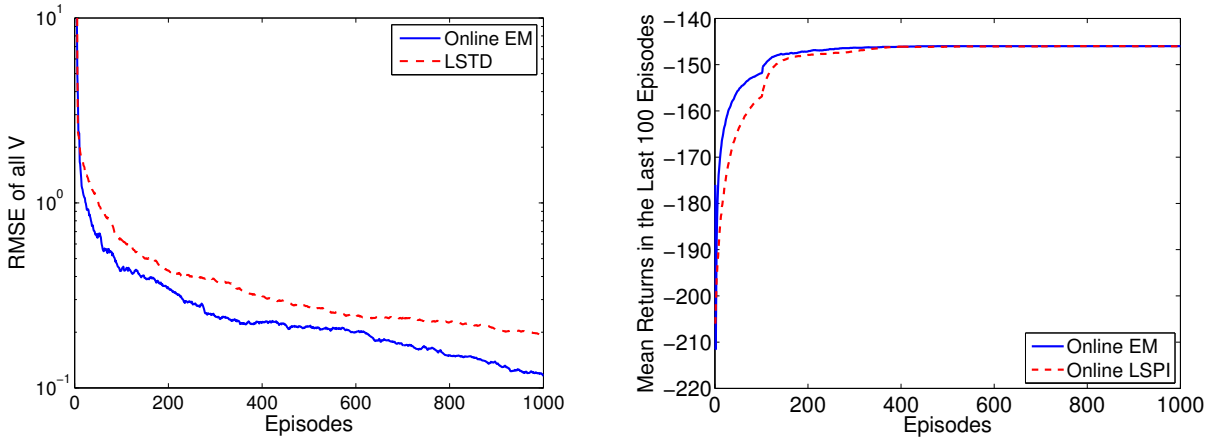
The results of the experiments for the Boyan's chain are illustrated in Fig. 3.2. More specifically, the Root Mean Square Error (RMSE) between the true and estimated value function is represented for policy evaluation problem, while the expected return of the last 100 episodes is used for evaluating the ability of online EM and online LSPI algorithms to discover an optimal policy (control problem). As it becomes apparent, the online EM algorithm achieves to approximate the value function of a given policy much more accurately in comparison with the standard LSTD algorithm. This happens as the proposed scheme approximates the model of the environment precisely, keeping explicit statistics. With regard to the control learning problem, both methodologies achieve to discover the optimal policies, with the proposed online EM scheme to converge at a higher rate than the online LSPI algorithm.

Fig. 3.3 shows the results of the experiments on the Puddle world benchmark. In

[2]More details can be found at: http://robots.mobilerobots.com/wiki/MobileSim



(a)13-states chain



(b) 98-states chain

Figure 3.2: Comparative results on policy evaluation and learned policy in Boyan's chain domain with 13 (a) and 98 (b) states. Each curve is the average of 30 independent trials.

the specific experiments, we evaluate the ability of the proposed scheme in discovering an optimal or near-optimal policy. In this context, we illustrate the expected return received by the agent during the last 30 episodes. The empirical results are in line with our prior belief that selecting an appropriate number of basis functions is a very important issue and that using a large number of RBFs does not imply better performance. All the above stem from the fact that the online EM algorithm manages to discover the optimal policy by creating approximately $450 - 550$ basis functions.

The comparative results on the real world environment are illustrated in Fig. 3.4. The first plot shows the number of episodes required by the robot so as to discover the target, per algorithm. Moreover, the second plot represents the expected return received by the agent during the last 100 episodes. In the case of LSPI algorithm, an equidistant, fixed 10×10 grid of Gaussian RBFs is used over the state spaces (800 RBFs in total). As it becomes clear, the online EM methodology achieves to discover

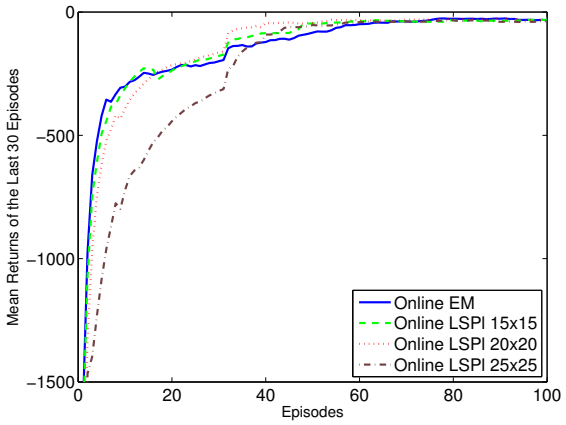


Figure 3.3: Comparative results in the Puddle World domain.

an optimal policy at a much higher rate, while at the same time it reaches the target in more episodes in contrast to the online LSPI method. Furthermore, our algorithmic scheme does not require a large number of basis functions in order to discover an optimal policy, as it happens in case of the online LSPI algorithm. More specifically, the online EM algorithm constructs approximately $300 - 400$ clusters that define the basis functions of our model. Finally, Fig. 3.5 represents the learned policies of both methods after 500 episodes. As we can observe, both algorithms tend to find the same policies in most world areas, while they achieve to reach the target starting from any position.

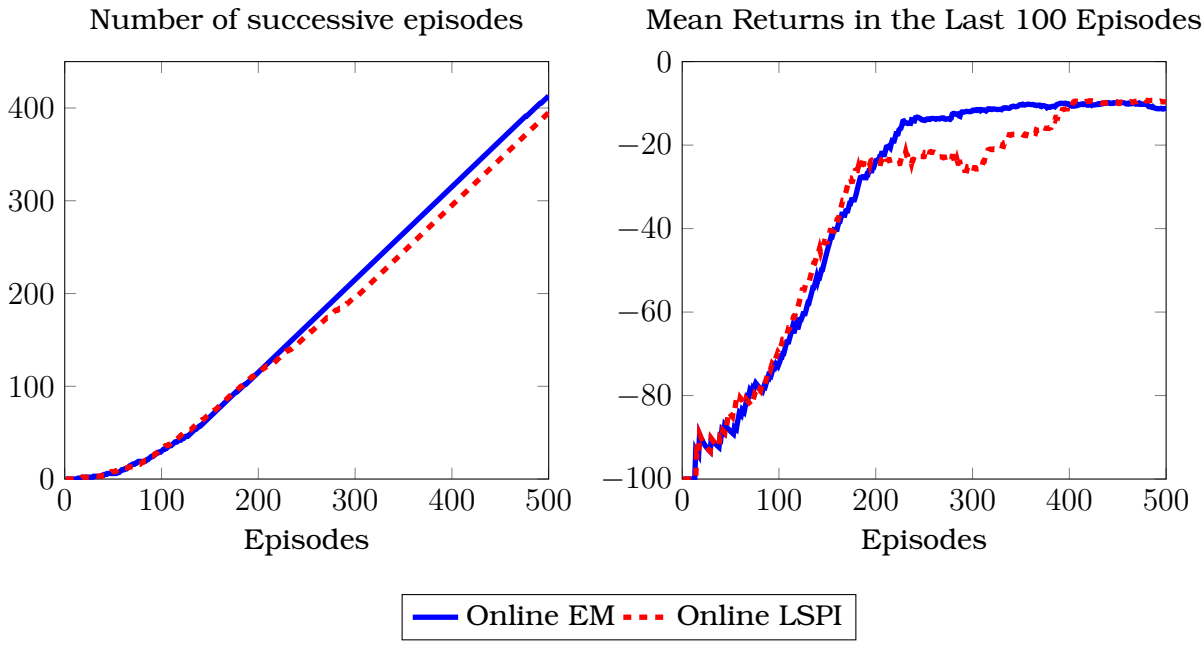
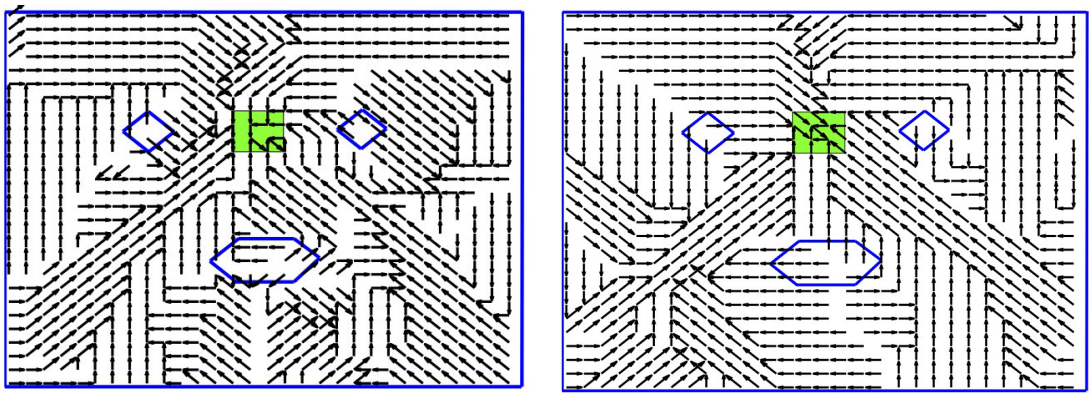


Figure 3.4: Comparative results in the real world domain.



(a) Online EM (b) Online LSPI

Figure 3.5: Learned policies by both comparative methods in the case of real world domain.

3.4 Summary

In this chapter, we have presented a model-based reinforcement learning scheme for both policy evaluation problem and control problem in unknown MDPs. The proposed algorithm is based on the online partitioning of the input (state-action) space into clusters, grouping data with similar properties. An appropriate mixture model has been adopted for this purpose, which is updated incrementally by using an online extension of the standard EM-algorithm. In this way, a number of basis functions are created and updated in an online mode. These basis functions are then used for the approximation of the value function, where the value function is formulated as a linear model. The adopted clustering scheme also allows us to keep in memory statistics about the model of the environment. Thus, the model of the environment is learnt as the agent wanders off inside it, and then it is used in the policy evaluation problem. The least-squares solution is used for the evaluation of the unknown coefficients of the value function model. In this way, the proposed scheme is able to estimate and update the policy followed by the agent at each time step. Experiments on both simulated and real environments demonstrate that the proposed algorithm performs equally well or better than standard reinforcement learning algorithms (e.g. LSPI), where a careful selection of the basis function must have been done in advance based on our prior environment knowledge. In contrast, our scheme achieves to discover the basis function automatically through its interaction with the environment.

Chapter 4

Linear Bayesian Reinforcement Learning

Bayesian reinforcement learning models the reinforcement learning problem as a decision-theoretic problem by placing a prior distribution p on the set of possible MDPs $\mathcal{M}$. However, the exact solution of the decision problem is generally intractable as the transformed problem becomes a Markov decision processes with exponentially many states [44].

This chapter focuses on Bayesian methods for solving the reinforcement learning problem (see [146] for an overview). This is a decision-theoretic approach [33], with two key ideas. The first is to select an appropriate prior distribution p about the unknown environment, such that $p(\mu)$ represents our subjective belief that μ is the true environment. The second is to replace the expected utility over the real environment, which is unknown, with the expected utility under the subjective belief p, i.e.

$$\mathbb{E}_p^\pi U = \int_{\mathcal{M}} \left(\mathbb{E}_\mu^\pi U\right) \, \mathrm{d}p(\mu), \tag{4.0.1}$$

where $U \triangleq \sum_{t=0}^{\infty} \gamma^t r_t$ is the agent's utility. Formally, it is then possible to optimise with respect to the policy which maximises the expected utility over all possible environments, according to our belief. However, our future observations will alter our future beliefs according to Bayes' theorem. In particular the posterior mass placed on a set of MDPs $B \subset \mathcal{M}$ given a history h_t composed of a sequence of states, $s^t = s_1, \ldots, s_t$, actions $a^{t-1} = a_1, \ldots, a_{t-1}$, is:

$$p(B \mid h_t) \triangleq \frac{\int_B \prod_{k=1}^t \frac{\mathrm{d}}{\mathrm{d}\nu} P_\mu(s_{t+1} \mid a_t, s_t) \, \mathrm{d}p(\mu)}{\int_{\mathcal{M}} \prod_{k=1}^t \frac{\mathrm{d}}{\mathrm{d}\nu} P_\mu(s_{t+1} \mid a_t, s_t) \, \mathrm{d}p(\mu)}, \tag{4.0.2}$$

where $\frac{\mathrm{d}}{\mathrm{d}\nu} P_\mu$ denotes the Radon-Nikodym derivative with respect to some measure ν on $\mathcal{S}$.[1] Consequently, the Bayes-optimal policy must take into account all potential future belief changes. For that reason, it will not in general be Markovian with respect to the states, but will depend on the complete history.

Most previous work on Bayesian reinforcement learning in continuous environments has focused on Gaussian process models for estimation. However, these suffer from two limitation. Firstly, they have significant computational complexity. Secondly, each dimension of the predicted state distribution is modeled independently. In this chapter, we investigate the use of Bayesian inference under the assumption that the dynamics are (perhaps under a transformation) linear. Then the modeling problem becomes multivariate Bayesian linear regression, for which we can calculate Eq. 4.0.2 efficiently online.

An other novelty of our approach in this context is that we do not simply use the common heuristic of acting as though the most likely or expected model is correct. Instead, generate a *sample model* from the posterior distribution. We then draw trajectories from the sampled model and collect simulated data which we use to obtain a policy. The policy is then executed in the real environment. This form of *Thompson sampling* is known to be a very efficient exploration method in bandit and discrete problems. We also show its efficacy for continuous domains.

The remainder of this chapter is organised as follows. Section 4.1 gives an overview of related work and our contribution. Section 4.2 formally introduces our approach, with a description of the inference model in Sec. 4.2.1, and the policy selection method used in Sec. 4.2.2. Finally, the details of the online and offline versions of our algorithms are detailed in Sec. 4.2.3. Experimental results are presented in Sec. 4.3 and we conclude with a discussion of future directions in Sec. 4.4.

4.1 Bayesian Reinforcement Learning and Our Contribution

One of the first and most interesting approaches for approximate Bayesian reinforcement learning is *Thompson sampling*, which is also used in the work presented in this chapter. The idea is to sample a model from the posterior distribution, calculate the optimal policy for the sampled model, and then follow it for some period [117]. Thompson sampling has been recently shown to perform very well both in theory and practice in bandit problems [68, 2]. Extensions and related models include Bayesian sparse sampling [148], which uses Thompson sampling to deal with node expansion in the tree search. Taking multiple samples from the posterior can be used to estimate upper and lower bounds on the Bayes-optimal value function, which can then be used for tree search algorithms [37, 39]. Multiple samples can also be used to create an augmented optimistic model [6, 28]; or they can be used to construct a better lower bound [40].

[1] In the discrete case we may simply use $\mathbb{P}_\mu(s_{t+1} \mid a_t, s_t)$.

Finally, multiple samples can also be combined via voting schemes [42]. Other approaches attempt to build optimistic models without sampling. For example [71] adds an exploration bonus to rewards, while [5] uses optimistic transition functions by constructing an augmented MDP in a Bayesian analogue of UCRL [62].

For continuous state spaces, most Bayesian approaches have focused on Gaussian process (GP) models [103, 65, 46, 105, 35]. There are two key ideas that set our method apart from this work. Firstly, GP models are typically employed independently for each state feature. In contrast, the model we use deals naturally with correlated state features – consequently, less data may be necessary. Secondly, we do not calculate policies using the expected transition dynamics of the environment, as this is known to have potentially bad effects [101, 40]. Instead, value functions and policies are calculated by *sampling* from the posterior distribution of environments. This also important for efficient exploration.

In this chapter, we propose a linear model-based Bayesian framework for reinforcement learning, for arbitrary state spaces $\mathcal{S}$ and for discrete action spaces $\mathcal{A}$ using Thompson sampling. First, we define a prior distribution on *linear dynamical models*, using a suitably chosen *basis*. Bayesian inference in this model is *fully closed form*, so that given a set of example trajectories it is easy to sample a model from the posterior distribution. For each such sample, we estimate the optimal policy. Since closed-form calculation of the optimal policy is not possible for general cost functions even with linear dynamics, we use Approximate Dynamic Programming (ADP, see [14] for an overview) with trajectories drawn from the sampled model. The resulting policy can then be applied to the real environment.

We experimented with two different ADP approaches for finding a policy for a given sampled MDP. Both are Approximate Policy Iteration (API) schemes, using a set of trajectories generated from the sampled MDP to estimate a sequence of value functions and policies. For the policy evaluation step, we experimented with fitted value iteration (FVI) [47] and Least-squares Temporal Differences (LSTD) [21].

In the case where we use LSTD, the approach can be seen as an online, Bayesian generalisation of Least-squares Policy Iteration (LSPI) [75]. Instead of performing LSTDQ on the empirical transition matrix, we perform a least-squares fit on a sample model drawn from the posterior. This fit can be very accurate by drawing a large amount of simulated trajectories in the sampled model.

We consider two applications of this approach. In the *offline* case, data is collected using a uniformly random policy. We then generate a model from the posterior distribution and calculate a policy for it, which is then evaluated in the real environment. In the *online* case, data is collected using policies generated from the sampled models, as in Thompson sampling. At the beginning each episode, a model is sampled from the posterior and the resulting policy is executed in the real environment. Thus, there is no separate data collection and evaluation phase. Our results show that this approach successfully finds optimal policies quickly and consistently both in the online and in

the offline case, and that it has a significant overall advantage over LSPI.

4.2 Linear Bayesian Reinforcement Learning

The model presented in this chapter uses Bayesian inference to estimate the environment dynamics. The assumption is that, with a suitable basis, these dynamics are linear with Gaussian noise. Unlike approaches using Gaussian processes, however, the next-state distribution is not modeled using a product distribution, i.e. we do not assume that the various components of the state are independent. In a further innovation, rather than using the expected posterior parameters, we employ sampling from the posterior distribution. For each sampled model, we then obtain an approximately optimal policy by using approximate dynamic programming, which is then executed in the real environment. This form of *Thompson sampling* [117, 128] allows us to perform efficient exploration, with the policies naturally becoming greedier as the posterior distribution converges.

4.2.1 The Predictive Model

In our model we assume that, for a state set $\mathcal{S}$ there exists a mapping, $\boldsymbol{\phi} : \mathcal{S} \to \mathcal{X}$, to a k-dimensional vector space $\mathcal{X}$ such that the transformed state at time t is $\boldsymbol{x}_t \triangleq \boldsymbol{\phi}(s_t)$. The next state s_{t+1} is given by the output of a function, $g : \mathcal{X} \times \mathcal{A} \to \mathcal{S}$, of the transformed state, the action and some additive noise:

$$s_{t+1} = g(\boldsymbol{x}_t, a_t) + \varepsilon_t. \tag{4.2.1}$$

In this work, we model the noise ε_t and the function g as a multivariate linear-Gaussian model. This is parameterized via a set of $k \times k$ *design* matrices $\{ A_i \mid i \in \mathcal{A} \}$, such that

$$g(\boldsymbol{x}_t, a_t) = A_{a_t} \boldsymbol{x}_t \tag{4.2.2}$$

and a set of *covariance* matrices $\{ V_i \mid i \in \mathcal{A} \}$ for the noise. Then, the next state distribution is:

$$s_{t+1} \mid \boldsymbol{x}_t = \boldsymbol{x}, a_t = i \sim \mathcal{N}(A_i \boldsymbol{x}, V_i). \tag{4.2.3}$$

In order to model our uncertainty with a (subjective) prior distribution p, we have to specify the model structure. In our model, we do not assume independence between the output dimensions, something which could potentially make inference difficult. Fortunately, in this particular case, a conjugate prior exists in the form of the *matrix-normal distribution* for A and the *inverse-Wishart* distribution for V. Given V_i, the distribution for A_i is matrix-normal, while the marginal distribution of V_i is inverse-Wishart. More specifically,

$$A_i \mid V_i = \widehat{V} \sim \mathcal{N}(A_i \mid M, C, \widehat{V}) \tag{4.2.4}$$

$$V_i \sim \mathcal{W}(V_i \mid W, n), \tag{4.2.5}$$

where $\mathcal{N}_i$ is the prior distribution on dynamics matrices conditional on the covariance and two prior parameters: M, which is the prior mean and C which is the prior output (dependent variable) covariance. Finally, $\mathcal{W}$ is the marginal prior on covariance matrices, which has an inverse-Wishart distribution with W and n. More precisely, the distributions are:

$$\mathcal{N}(A_i \mid M, C, \widehat{V}) \propto e^{-\frac{1}{2}\operatorname{tr}\left[(A_i - M)^\top V_i^{-1}(A_i - M)C\right]}, \tag{4.2.6}$$

$$\mathcal{W}(V_i \mid W, n) \propto |V^{-1}W/2|^{n/2} e^{-\frac{1}{2}\operatorname{tr}(V^{-1}W)}. \tag{4.2.7}$$

Essentially, the model is an extension of the univariate Bayesian linear regression model (see for example [33]) to the multivariate case via vectorisation of the mean matrix. Since the prior is conjugate, it is relatively simple to calculate posterior values of the parameters after each observation. While we omit the details, a full description of inference using this model is given in [87].

Throughout this text, we shall employ $p_t = (\mathcal{N}_t, \mathcal{W}_t)$ to denote our posterior distributions at time t, with p_t referring to our complete posterior. The remaining problem is how to estimate the Bayes-expected utility of the current policy and how to perform policy improvement.

4.2.2 Policy Evaluation and Optimisation

In the Bayesian setting, policy evaluation and optimisation are not trivial. The most common method used is the expected MDP heuristic, where policies are evaluated or optimised on the expected MDP. However, this ignores the shape of the posterior distribution. Alternatively, policies can be evaluated via Monte Carlo sampling, but then optimisation becomes hard. A good heuristic that does not ignore the complete posterior distribution and for which it is easy to calculate a policy, called Thompson sampling, is the one we shall actually employ in our scheme. The following paragraphs give a quick overview of each method.

Expected MDP

A naive way to estimate the expected utility of a policy is to first calculate the expected (or most probable) dynamics, and then use either an exact or an approximate dynamic programming algorithm. This may very well be a good idea if the posterior distribution is sharply concentrated around the mean, since then:

$$\mathbb{E}_p^\pi U \approx \mathbb{E}_{\mu_p}^\pi U, \qquad \mu_p \triangleq \mathbb{E}_p \mu. \tag{4.2.8}$$

where μ_p is the expected MDP model.[2] However, as pointed out in [5, 40] this approach may give completely incorrect results.

[2] Similar problems exist when using the most probable MDP instead.

Monte Carlo Sampling

An alternative method is to take a number of samples μ_i from the current posterior distribution and then calculate the expected utility of each, i.e.

$$\mathbb{E}_p^\pi U = \frac{1}{K} \sum_{i=1}^{K} \mathbb{E}_{\mu_i}^\pi U + O(K^{-1/2}), \qquad \mu_i \sim p_t. \tag{4.2.9}$$

This form of Monte Carlo sampling gives much more accurate results, at the expense of some additional computation. However, finding an optimal policy over a set of sampled MDPs is difficult even for restricted classes of policies [40]. Nevertheless, Monte Carlo sampling can also be used to obtain stochastic upper and lower bounds on the value function, which can be used to improve the policy search [39, 37].

Thompson Sampling

An interesting special case is when we only sample a single MDP, i.e. when we perform Monte Carlo sampling with $K = 1$. Then it is relatively easy to calculate the optimal policy for this sample. This method, which we employ in this work, is called *Thompson sampling*, and was first used in the context of reinforcement learning by [117]. The idea is to sample an MDP from the current posterior and then calculate a policy that is optimal with respect to that MDP. We then execute this policy in the environment. The major advantage of Thompson sampling is that it is known to result in a very efficient form of exploration (see for example [2] for recent results on bandit problems).

4.2.3 Algorithm Overview

We can now put everything together for the complete linear Bayesian reinforcement learning (LBRL) algorithm. The algorithm has four steps. Firstly, sampling a model from the posterior distribution. Secondly, using the sampled model to calculate a new policy. Finally, executing this policy in the real environment. In the online version of the algorithm the data obtained by executing this policy is then used to calculate a new posterior distribution.

Sampling from the Posterior

Our posterior distribution at time t is $p_t = (\mathcal{N}_t, \mathcal{W}_t)$, with $\mathcal{W}_t$ being the marginal posterior on covariance matrices, and $\mathcal{N}_t$ being the posterior on design matrices (conditional on the covariance). In order to sample a model from the posterior, we first draw a covariance matrix V_i using Eq. 5.2.4 for every action $i \in \mathcal{A}$, and then plug those into Eq. 5.2.3 to generate a set of design matrices A_i. The first step requires sampling from the inverse-Wishart distribution (which can be done efficiently using the algorithm suggested by [115]), and the second from the matrix-normal distribution.

Algorithm 4.1: LBRL: Linear Bayesian Reinforcement Learning

Input: Basis $\boldsymbol{\phi}$, ADP parameters P, prior p_0

for *episode* k **do**
  $\mu_k \sim p_{t_k}(\mu)$; // Generate MDP from posterior
  $\pi_k = \texttt{ADP}(\mu_k, P)$; // Get new policy
  for $t = t_k, \ldots, t_{k+1} - 1$ **do**
    $a_t \mid s_t = s \sim \pi^{(k)}(a \mid s)$; // Take action
    $p_{t+1}(\mu) = p_t(\mu \mid s_{t+1}, a_t, s_t)$; // Update posterior
  end
end

ADP on the Sampled MDP

Given an MDP μ sampled from our posterior belief, we can calculate a nearly-optimal policy π using approximate dynamic programming (ADP) on μ. This can be done with a number of algorithms. Herein, we investigated two approximate policy iteration (API) schemes, using either fitted value iteration (FVI) or least-squares temporal differences (LSTD) for the policy evaluation step. Both of these algorithms require sample trajectories from the environment. This is fortunately very easy to achieve, since we can use the sampled model to generate any number of trajectories arbitrarily. Consequently, we can always have enough *simulated* data to perform a good fit with FVI or LSTD.[3] We note here that API using LSTD additionally requires a generative model for the policy improvement step. Happily, we can use the sampled MDP μ for that purpose.[4]

Offline LBRL

In the offline version of the algorithm, we simply collect a set of trajectories from a *uniformly random policy*, comprising a history h_t of length t. Then, we sample an MDP from the posterior $p_t(\mu) = p_0(\mu \mid h_t)$ and calculate the optimal policy for the sample using ADP. This policy is then *evaluated* on the real environment.

Online LBRL

In the online version of the algorithm, shown in Alg. 4.1 we collect samples using our *own generated policies*. We begin with some initial belief $p_0 = (\mathcal{N}_0, \mathcal{W}_0)$ and a uniformly random policy π_0. This policy is executed until either the episode ends naturally or due to reaching a time-limit T. At the k-th episode, which starts at time t_k, we sample a new MDP $\mu_k \sim p_{t_k}$ from our current posterior $p_{t_k}(\cdot) = p(\cdot \mid h_{t_k})$ and then calculate a

[3] These use no data collected in the real environment.

[4] In preliminary experiments, we also investigated the use of fitted Q-iteration and LSPI, but found that these had inferior performance.

near-optimal stationary policy for $\mu^{(k)}$:

$$\pi_k \approx \arg\max_{\pi} \mathbb{E}^{\pi}_{\mu^{(k)}} U,$$

such that $\pi_k(a \mid s)$ is a conditional distribution on actions $a \in \mathcal{A}$ given states $s \in \mathcal{S}$. This policy is then executed in the real environment until the end of the episode and the data collected are used to calculate the new posterior. As the calculation of posterior parameters is fully incremental, we incur no additional computational cost for running this algorithm online.

4.3 Empirical Evaluation

We conducted two sets of experiments to analyse both the offline and the online performance of the various algorithms. Comparisons have been made with the well-known Least-Squares Policy Iteration (LSPI) algorithm [75] for the offline case, as well as an online variant [27] for the online case. We used preliminary runs and guidance from the literature to select the features for the LSTDQ algorithm used in the inner loop of LSPI. The source for all the experiments can be found in [41].

We employed the same features for the ADP algorithms used in LBRL. However, the basis used for the Bayesian regression model in LBRL was simply $\boldsymbol{\phi}(\boldsymbol{s}) \triangleq [\boldsymbol{s}, 1]^{\top}$. In preliminary experiments, we found this sufficient for a high-quality approximation. After that, we use API to find a good policy for a sampled MDP, where we experimented with regularised FVI and LSTD for the policy evaluation step, adding a regularisation factor $10^{-2}I$. In both cases, we drew single step transitions from a set of 3000 uniformly drawn states from the sampled model.

For the offline performance evaluation, we first drew rollouts from $k = \{50, 100, \ldots, 1000\}$ states drawn from the environment's starting distribution, using a uniformly random policy. The maximum horizon of each rollout was set equal to 40. The collected data was then fed to each algorithm in order to produce a policy. This policy was evaluated over 1000 rollouts on the environment.

In the online case, we simply use the last policy calculated by each algorithm at the end of the last episode, so there is no separate learning and evaluation phase. This means that efficient exploration must be performed. For LBRL, this is done using Thompson sampling. For online-LSPI, we followed the approach of [27], who adopts an ϵ-greedy exploration scheme with an exponentially decaying schedule $\epsilon_t = \epsilon_d^t$, with $\epsilon_0 = 1$. In preliminary experiments, we found $\epsilon_d = 0.9968$ to be a reasonable compromise. We compared the algorithms online for 1000 episodes.

4.3.1 Domains

We consider two well-known continuous state, discrete-action, episodic domains. The first is the *mountain car* domain and the second is *inverted pendulum* the domain.

Mountain Car

In the second experimental set, we have used the *mountain car* environment. Two continuous variables characterise the vehicle state in the domain, its position (p) and its velocity (u). The objective in this task is to drive an underpowered vehicle up a steep road from a randomly selected position to the right hilltop ($p \geq 0.5$) with at most 1000 steps. In order to achieve our goal, we can select between three actions: forward, reverse and zero throttle. The received reward is $r = -1$ except in the case where the target is reached (zero reward). At the beginning of each rollout, the vehicle is positioned to a new state, with the position and the velocity uniformly randomly selected. The discount factor is equal to 0.999. An equidistant 4×4 grid of RBFs over the state space plus a constant term is selected for FVI/LSTD and LSPI.

Inverted Pendulum

The first set of experiments includes the *inverted pendulum* domain, which tries to balance a pendulum by applying forces of a fixed magnitude (50 Newtons). The state space consists of two continuous variables, the vertical angle (θ) and the angular velocity ($\dot{\theta}$) of the pendulum. The agent has at his arsenal three actions: no force, left force or right force. A zero reward is received at each time step except in the case where the pendulum falls ($|\theta| \leq \pi/\ 2$). In this case, a negative ($r = -1$) reward is given and a new rollout begins. Each rollout starts by setting the pendulum in a perturbed state close to the equilibrium point. More information about the environment dynamics can be found at [75]. Each rollout is allowed to run for 3000 steps at maximum. Additionally, the discount factor is set to $\gamma = 0.95$. For FVI/LSTD and LSPI, we used an equidistant 3×3 grid of RBFs over the state space following the suggestions of [75], which was replicated for each action for the LSTDQ algorithm used in LSPI.

4.3.2 Results

In our results, we show the average performance in terms of number of steps of each method, averaged over 100 runs. For each average, we also plot the 95% confidence interval for the accuracy of the mean estimate with error bars. In addition, we show the 90% percentile region of the runs, in order to indicate inter-run variability in performance.

Figure 5.3(a) shows the results of the experiments in the *offline* case. For the *mountain car*, it is clear that the most stable approach is LBRL-LSTD, while LSPI is the most unstable. Nevertheless, on average the performance of LBRL-LSTD and LSPI is similar, while LBRL-FVI is slightly worse. For the *pendulum* domain, the performance of LBRL remains quite good, with LBRL-LSTD being the most stable. While LSPI manages to find the optimal policy frequently, nevertheless around 5% of its runs fail.[5]

[5]We note that the results presented in [75] for LSPI are slightly better, but remain significantly below the LBRL results.

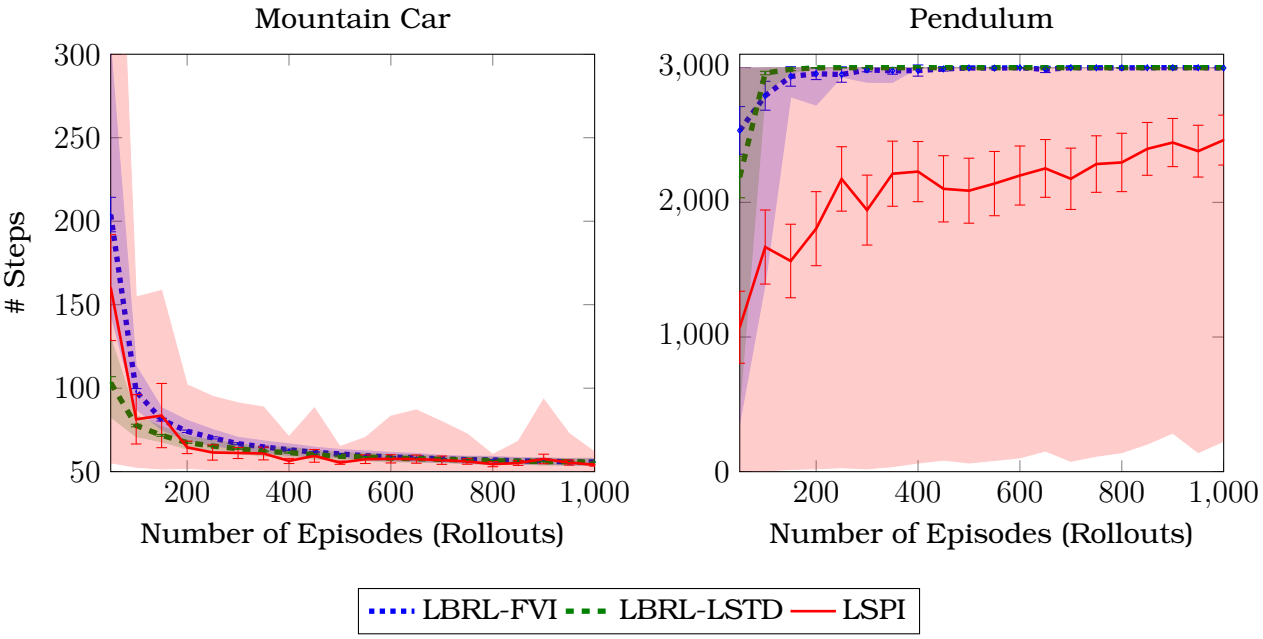


Figure 4.1: Offline performance comparison between LBRL-FVI, LBRL-LSTD and LSPI. The error bars show 95% confidence intervals, while the shaded regions show 90% percentiles over 100 runs.

Figure 5.3(b) shows the results of the experiments in the *online* case. For the *mountain car*, both LSPI and LBRL-LSTD managed to find an excellent policy in the vast majority of runs. In the *pendulum* domain, we see that LBRL-LSTD significantly outperforms LSPI. In particular, after 80 episodes all more than 90% of the runs are optimal, while many LSPI runs fail to find a good solution even after hundreds of episode. The mean difference is somewhat less spectacular, though still significant.

The success of LBRL-LSTD over LSPI can be attributed to a number of reasons. Firstly, it could be the *more efficient exploration*. Indeed, in the mountain car domain, where the starting state distribution is uniform, we can see that LBRL and LSPI have very similar performance. Another possible reason is that LBRL also makes *better use of the data*, since it uses it to calculate the posterior distribution over MDP dynamics. It is then possible to perform very accurate ADP using simulated data from a model sampled from the posterior. This is supported by the offline results in the pendulum domain.

4.4 Summary

In this chapter, we have presented a simple linear Bayesian approach to reinforcement learning in continuous domains. Unlike Gaussian process models, by using a linear-Gaussian model, we have the potential to scale up to real world problems which Bayesian reinforcement learning usually fails to solve with a reasonable amount of

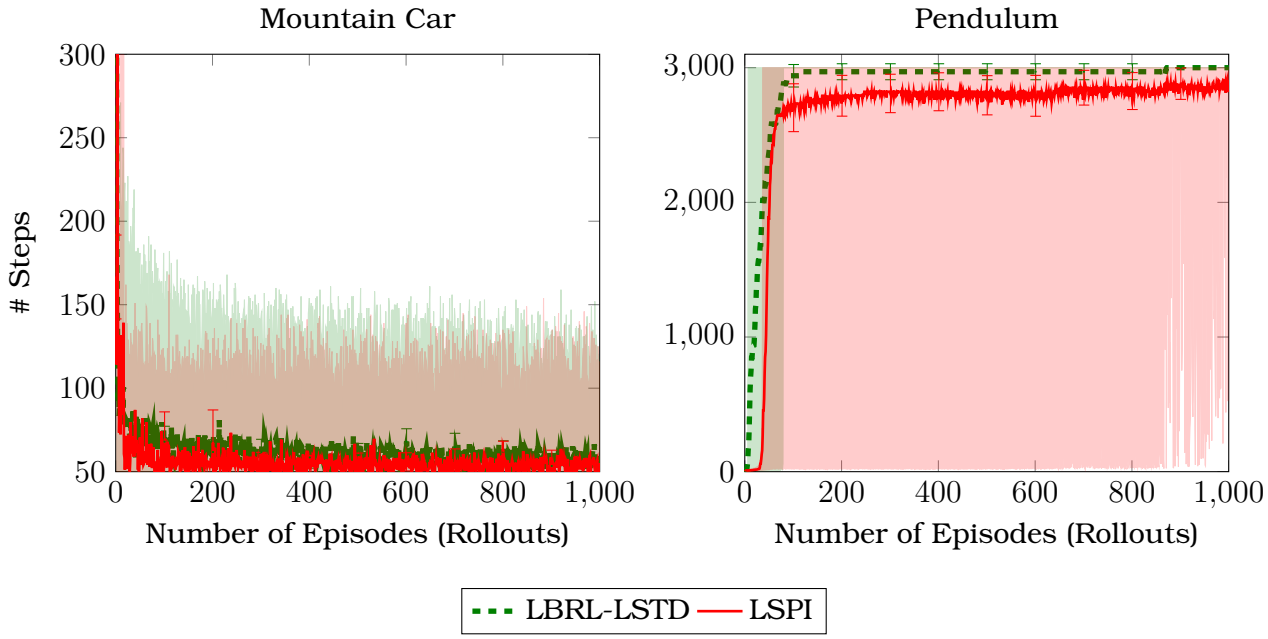


Figure 4.2: Online performance comparison between LBRL-LSTD and LSPI. The error bars show 95% confidence intervals, while the shaded regions show 90% percentiles over 100 runs.

data. In addition, this model easily takes into account correlations in the state features, further reducing sample complexity. We solve the problem of computing a good policy in continuous domains with uncertain dynamics by using Thompson sampling. This not much more expensive than computing the expected MDP and forces a natural exploration behaviour.

In practice, the algorithm is at least as good as LSPI in offline mode, while being considerably more stable overall. When LBRL is used to perform online exploration, we find that the algorithm very quickly converges to a near-optimal policy and is extremely stable. Experimentally, it would be interesting to compare LBRL with standard GP methods that employ the expected MDP heuristic.

Chapter 5

Cover Tree Bayesian Reinforcement Learning

Efficient learning and planning requires models of the environment that are not only general, but can also be updated online with low computational cost. In addition, probabilistic models allow the use of a number of near-optimal algorithms for decision making under uncertainty. While it is easy to construct such models for small, discrete environments, models for the continuous case have so far been mainly limited to parametric models, which may not have the capacity to represent the environment (such as generalised linear models) and to non-parametric models, which do not scale very well (such as Gaussian processes).

In this chapter, we propose a non-parametric family of tree models, with a data-dependent structure constructed through the cover tree algorithm, introduced by [16]. Cover trees are data structures that cover a metric space with a sequence of data-dependent partitions. They were initially proposed for the problem of k-nearest neighbour search, but they are in general a good method to generate fine partitions of a state space, due to their low complexity, and can be applied to any state space, with a suitable choice of metric. In addition, it is possible to create a statistical model using the cover tree as a basis. Due to the tree structure, online inference has low (logarithmic) complexity.

We specifically investigate the case of a Euclidean state space. For this, we propose a model generalising the context tree weighting algorithm proposed by [152], combined

with Bayesian multivariate linear models. The overall prior can be interpreted as a distribution on piecewise-linear models. We then compare this model with a Gaussian process model, a single linear model, and the model-free method least-squares policy iteration in two well-known benchmark problems in combination with approximate dynamic programming and show that it consistently outperforms other approaches.

The remainder of the chapter is organised as follows. Section 5.1 introduces the general setting of the Bayesian framework for reinforcement learning. Section 5.1.1 discusses the context tree notion and other related work. The model and algorithm are described in Section 5.2. Finally, comparative experiments are presented in Section 5.3 and we conclude with a discussion of the advantages of cover-tree Bayesian reinforcement learning and directions of future work in Section 5.4.

5.1 Bayesian Framework in Unknown MDPs

We assume that the agent acts within a fully observable discrete-time Markov decision process (MDP), with a metric state space $\mathcal{S}$, for example $\mathcal{S} \subset \mathbb{R}^m$. At time t, the agent observes the current environment state $s_t \in \mathcal{S}$, takes an action a_t from a discrete set $\mathcal{A}$, and receives a reward $r_t \in \mathbb{R}$. The probability over next states is given in terms of a transition kernel $P_\mu(S \mid s, a) \triangleq \mathbb{P}_\mu(s_{t+1} \in S \mid s_t = s, a_t = a)$. The agent selects its actions using a *policy* $\pi \in \Pi$, which in general defines a conditional distribution $\mathbb{P}^\pi(a_t \mid s_1, \ldots, s_t, a_1, \ldots, a_{t-1}, r_1, \ldots, r_{t-1})$ over the actions, given the history of states and actions. This reflects the learning process that the agent undergoes, when the MDP μ is unknown.

The agent's *utility* is $U \triangleq \sum_{t=0}^{\infty} \gamma^t r_t$, the discounted sum of future rewards, with $\gamma \in (0, 1)$ a discount factor such that rewards further into the future are less important than immediate rewards. The goal of the agent is to maximise its expected utility:

$$\max_{\pi \in \Pi} \mathbb{E}_\mu^\pi U = \max_{\pi \in \Pi} \mathbb{E}_\mu^\pi \sum_{t=0}^{\infty} \gamma^t r_t,$$

where the value of the expectation depends on the agent's policy π and the environment μ. If the environment is known, well-known dynamic programming algorithms can be used to find the optimal policy in the discrete-state case [102], while many approximate algorithms exist for continuous environments [15]. In this case, optimal policies are memoryless and we let Π_1 denote the set of memoryless policies. Then MDP and policy define a Markov chain with kernel $P_\mu^\pi(S \mid s, a) = \sum_{a \in A} P_\mu(S \mid s, a)\pi(a \mid s)$.

However, since the environment μ is unknown, the above maximisation is ill-posed. In the Bayesian framework for reinforcement learning, this problem is alleviated by performing the maximisation conditioned on the agent's belief about the true environment μ. This converts the problem of reinforcement learning into a concrete, optimisation problem. However, this is generally extremely complex, as we must optimise over all history-dependent policies.

More specifically, the main assumption in Bayesian reinforcement learning is that the environment μ lies in a given set of environments $\mathcal{M}$. In addition, the agent must select a subjective prior distribution $p(\mu)$ which encodes its belief about which environments are most likely. The Bayes-optimal expected utility for p is:

$$U_p^* \triangleq \max_{\pi \in \Pi^D} \mathbb{E}_p^\pi U = \max_{\pi \in \Pi^D} \int_{\mathcal{M}} \left(\mathbb{E}_\mu^\pi U \right) \mathrm{d}p(\mu). \tag{5.1.1}$$

Unlike the known μ case, the optimal policy may not be memoryless, as our belief changes over time. This makes the optimisation over the policies significantly harder [44], as we have to consider the set of all history-dependent deterministic policies, which we denote by $\Pi^D \subset \Pi$. In this chapter, we employ the simple, but effective, heuristic of Thompson sampling [128, 154, 32, 117] for finding policies. This strategy is known by various other names, such as probability matching, stochastic dominance, sampling-greedy and posterior sampling. Very recently [92] showed that it suffers small Bayes-regret relative to the Bayes-optimal policy for finite, discrete MDPs.

The second problem in Bayesian reinforcement learning is the choice of the prior distribution. This can be of critical importance for large or complex problems, for two reasons. Firstly, a well-chosen prior can lead to more efficient learning, especially in the finite-sample regime. Secondly, as reinforcement learning involves potentially unbounded interactions, the computational and space complexity of calculating posterior distributions, estimating marginals and performing sampling become extremely important. The choice of priors is the main focus of the work proposed in this chapter. In particular, we introduce a prior over piecewise-linear multivariate Gaussian models. This is based on the construction of a context tree model, using a cover tree structure, which defines a conditional distribution on local linear Bayesian multivariate models. Since inference for the model can be done in closed form, the resulting algorithm is very efficient, in comparison with other non-parametric models such as Gaussian processes. The following section discusses how previous work is related to our model.

5.1.1 Context Trees Inference

One component in our model is the *context tree*. Context trees were introduced by [152] for sequential prediction (see [11], for an overview). In this model, a distribution of variable order Markov models for binary sequences is constructed, where the tree distribution is defined through context-dependent weights (for probability of a node being part of the tree) and Beta distributions (for predicting the next observation). A recent extension to switching time priors [142] has been proposed by [143]. More related to this work is an algorithm proposed by [74] for prediction. This asymptotically converges to the best univariate piecewise linear model in a class of trees with fixed structure.

Many reinforcement learning approaches based on such trees have been proposed, but have mainly focused on the discrete partially observable case [31, 144, 12, 48].[1]

[1]We note that another important work in tree-based reinforcement learning, though not directly

However, tree structures can generally be used to perform Bayesian inference in a number of other domains [93, 84, 153].

The core of our model is a generalised context tree structure that defines a distribution on multivariate piecewise-linear-Gaussian models. Consequently, a necessary component in our model is a multivariate linear model at each node of the tree. Such models were previously used for Bayesian reinforcement learning in [137] and were shown to perform well relatively to Least-Squares Policy Iteration (LSPI, [75]). Other approaches using linear models include [116], which proves mistake bounds on reinforcement learning algorithms using online linear regression, and [1] who use separate linear models for each dimension. Another related approach in terms of structure is [24], which partitions the space into *types* and estimates a simple additive model for each type.

Linear-Gaussian models are naturally generalised by Gaussian processes (GP). Some examples of GP in reinforcement learning include those of [103], [35] and [34], which focused on a model-predictive approach, while the work of [46] employed GPs for expected utility estimation. GPs are computationally demanding, in contrast to our tree-structured prior. Another problem with the cited GP-RL approaches is that they employ the marginal distribution in the dynamic programming step. This heuristic ignores the uncertainty about the model (which is implicitly taken into account in Equations 5.1.1, 5.2.5). A notable exception to this is the policy gradient approach employed by [56] which uses full Bayesian quadrature. Finally, output dimensions are treated independently, which may not make good use of the data. Methods for efficient dependent GPs such as the one introduced by [4] have not yet been applied to reinforcement learning. For decision making, our approach uses the simple idea of Thompson sampling [128, 154, 32, 117], which has been shown to be near-optimal in certain settings [68, 2, 92]. This avoids the computational complexity of building augmented MDP models [8, 6, 28, 5], Monte-Carlo tree search [144], sparse sampling [148], stochastic branch and bound [39] or creating lower bounds on the Bayes-optimal value function [101, 40]. Thus the approach is reasonable as long as sampling from the model is efficient.

5.2 Cover Tree Bayesian Reinforcement Learning

The main idea of Cover Tree Bayesian Reinforcement Learning (CTBRL) is to construct a cover tree from the observations, simultaneously inferring a conditional probability density on the same structure, and to then use sampling to estimate a policy. We use a cover tree due to its efficiency compared with e.g., a fixed sequence of partitions or other dynamic partitioning methods such as KD-trees. The probabilistic model we use can be seen as a distribution over piecewise linear-Gaussian densities, with one local

related to ours, is that of [47], which uses trees for expected utility rather than model estimation.

linear model for each set in each partition. Due to the tree structure, the posterior can be computed efficiently online. By taking a sample from the posterior, we acquire a specific piecewise linear Gaussian model. This is then used to find an approximately optimal policy using approximate dynamic programming.

An overview of CTBRL is given in pseudocode in Alg. 5.1. As presented, the algorithm works in an episodic manner.[2] When a new episode k starts at time t_k, we calculate a new stationary policy by sampling a tree μ_k from the current posterior $p_{t_k}(\mu)$. This tree corresponds to a piecewise-linear model. We draw a large number of rollout trajectories from μ_k using an arbitrary exploration policy. Since we have the model, we can use an initial state distribution that covers the space well. These trajectories are used to estimate a near-optimal policy π_k using approximate dynamic programming. During the episode, we take new observations using π_k, while growing the cover tree as necessary and updating the posterior parameters of the tree and the local model in each relevant tree node.

Algorithm 5.1: CTBRL (Episodic, using Thompson Sampling)

Initialize: $k = 0$, $\pi_0 = \mathcal{U}nif(\mathcal{A})$, prior p_0 on $\mathcal{M}$.

1 **for** $t = 1, \ldots, T$ **do**
2 **if** `episode-end` **then**
3 $k := k + 1$.
4 Sample model $\mu_k \sim p_t(\mu)$.
5 Calculate policy $\pi_k \approx \arg\max_\pi \mathbb{E}^\pi_{\mu_k} U$.
6 **end**
7 Observe state s_t.
8 Take action $a_t \sim \pi_k(\cdot \mid s_t)$.
9 Observe next state s_{t+1}, reward r_{t+1}.
10 Add a leaf node to the tree $\mathcal{T}_{a_t}$, containing s_t.
11 Update posterior: $p_{t+1}(\mu) = p_t(\mu \mid s_{t+1}, s_t, a_t)$ by updating the parameters of all nodes containing s_t.
12 **end**

We now explain the algorithm in detail. First, we give an overview of the cover tree structure on which the context tree model is built. Then we show how to perform inference on the context tree, while Section 5.2.3 describes the multivariate model used in each node of the context tree. The sampling approach and the approximate dynamic method are described in Section 5.2.4, while the overall complexity of the algorithm is discussed in Section 5.2.5.

[2] An online version of the same algorithm (still employing Thompson sampling) would move line 5 to just before line 8. A fully Bayes online version would "simply" take an approximation of the Bayes-optimal action at every step.

5.2.1 The Cover Tree Structure

Cover trees are a data structure that can be applied to any metric space and are, among other things, an efficient method to perform nearest-neighbour search in high-dimensional spaces [16]. In the particular work, we use cover trees to automatically construct a sequence of partitions of the state space. Section 5.2.1 explains the properties of the constructed cover tree. As the formal construction duplicates nodes, in practice we use a reduced tree where every observed point corresponds to one node in the tree. This is explained in Section 5.2.1. An explanation of how nodes are added to the structure is given in Section 5.2.1.

Cover Tree Properties

To construct a cover tree $\mathcal{T}$ on a metric space $(\mathcal{Z}, \psi)$ we require a set of points $D_t = \{z_1, \ldots, z_t\}$, with $z_i \in \mathcal{Z}$, a metric ψ, and a constant $\zeta > 1$. We introduce a mapping function $[\cdot]$ so that the i-th tree node corresponds to one point $z_{[i]}$ in this set. The nodes are arranged in *levels*, with each point being replicated at nodes in multiple levels, i.e., we may have $[i] = [j]$ for some $i \neq j$. Thus, a point corresponds to multiple nodes in the tree, but to *at most one node* at any one level. Let G_n denote the set of points corresponding to the nodes at level n of the tree and $\mathfrak{C}(i) \subset G_{n-1}$ the corresponding set of children. If $i \in G_n$ then the level of i is $\ell(i) = n$. The tree has the following properties:

1. Refinement: $G_n \subset G_{n-1}$.
2. Siblings separation: $i, j \in G_n$, $\psi(\mathbf{z}_{[i]}, \mathbf{z}_{[j]}) > \zeta^n$.
3. Parent proximity: If $i \in G_{n-1}$ then $\exists$ a unique $j \in G_n$ such that $\psi(\mathbf{z}_{[i]}, \mathbf{z}_{[j]}) \leq \zeta^n$ and $i \in \mathfrak{C}(j)$.

These properties can be interpreted as follows. Firstly lower levels always contain more points. Secondly, siblings at a particular level are always well-separated. Finally, a child must be close to its parent. These properties directly give rise to the theoretical guarantees given by the cover tree structure, as well as methods for searching and adding points to the tree, as explained below.

The Reduced Tree

As formally the cover tree duplicates nodes, in practice we use the *explicit representation* (described in more detail in Section 2 of [16]). This only stores the top-most tree node i corresponding to a point $z_{[i]}$. We denote this *reduced tree* by $\hat{\mathcal{T}}$. The *depth* $d(i)$ of node $i \in \hat{\mathcal{T}}$ is equal to its number of ancestors, with the root node having a depth of 0. After t observations, the set of nodes containing a point z, is:

$$\hat{G}_t(z) \triangleq \left\{ i \in \hat{\mathcal{T}} \;\middle|\; z \in B_i \right\},$$

where $B_i = \left\{ z \in \mathcal{Z} \;\middle|\; \psi(z_{[i]}, z) \leq \zeta^{d(i)} \right\}$ is the neighbourhood of i. Then $\hat{G}_t(z)$ forms a path in the tree, as each node only has one parent, and can be discovered in logarithmic time through the `Find-Nearest` function (Theorem 5, [16]). This fact allows us to efficiently search the tree, insert new nodes, and perform inference.

Inserting Nodes in the Cover Tree

The cover tree insertion we use is only a minor adaptation of the `Insert` algorithm by [16]. For each action $a \in \mathcal{A}$, we create a different reduced tree $\hat{\mathcal{T}}_a$, over the state space, i.e., $\mathcal{Z} = \mathcal{S}$, and build the tree using the metric $\psi(s, s') = \|s - s'\|_1$.

At each point in time t, we obtain a new observation tuple s_t, a_t, s_{t+1}. We select the tree $\hat{\mathcal{T}}_{a_t}$ corresponding to the action. Then, we traverse the tree, decreasing d and keeping a set of nodes $Q_d \subset G_d$ that are ζ^d-close to s_t. We stop whenever Q_d contains a node that would satisfy the *parent proximity* property if we insert the new point at $d - 1$, while the children of all other nodes in Q_d would satisfy the *sibling separation* property. This means that we can now insert the new datum as a child of that node.[3] Finally, the next state s_{t+1} is only used during the inference process, explained below.

5.2.2 Generalised Context Tree Inference

In our model, each node $i \in \hat{\mathcal{T}}$ is associated with a particular Bayesian model. The main problem is how to update the individual models and how to combine them. Fortunately, a closed form solution exists due to the tree structure. We use this to define a *generalised context tree*, which can be used for inference.

As with other tree models [152, 50], our model makes predictions by marginalising over a set of simpler models. Each node in the context tree is called a *context*, and each context is associated with a specific local model. At time t, given an observation $s_t = s$ and an action $a_t = a$, we calculate the marginal (predictive) density p_t of the next observation:

$$p_t(s_{t+1} \mid s_t, a_t) = \sum_{c_t} p_t(s_{t+1} \mid s_t, c_t) p_t(c_t \mid s_t, a_t),$$

where we use the symbol p_t throughout for notational simplicity to denote marginal distributions from our posterior at time t. Here, c_t is such that if $p_t(c_t = i \mid s_t, a_t) > 0$, then the current state is within the neighbourhood of i-th node of the reduced cover tree $\hat{\mathcal{T}}_{a_t}$, i.e., $s_t \in B_i$.

For Euclidean state spaces, the i-th component density $p_t(s_{t+1} \mid s_t, c_t = i)$ employs a linear Bayesian model, which we describe in the next section. The graphical structure of the model is shown in simplified form in Fig. 5.1. The context at time t depends only on the current state s_t and action a_t. The context corresponds to a particular local model with parameter ϑ_t, which defines the conditional distribution.

[3] The exact implementation is available in the CoverTree class in [41].

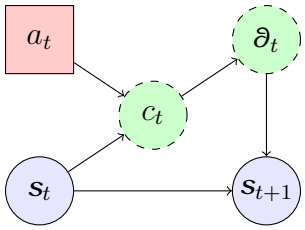


Figure 5.1: The generalised context tree graphical model. Blue circles indicate observed variables. Green dashed circles indicate latent variables. Red rectangles indicate choice variables. Arrows indicate dependencies. Thus, the context distribution at time t depends on both the state and action, while the parameters depend on the context. The next state depends on the action only indirectly.

The probability distribution $p_t(c_t \mid \boldsymbol{s}_t, a_t)$ is determined through *stopping probabilities*. More precisely, we set it be equal to the probability of stopping at the i-th context, when performing a walk from the leaf node containing the current observation towards the root, stopping at the j-th node with probability $w_{j,t}$ along the way:

$$p_t(c_t = i \mid \boldsymbol{s}_t, a_t) = w_{i,t} \prod_{j \in \mathfrak{D}_t(i)} (1 - w_{j,t}),$$

where $\mathfrak{D}_t(i)$ are the descendants of i that contain the observation $\boldsymbol{s}_t$. This forms a path from i to the leaf node containing $\boldsymbol{s}_t$. Note that $w_{0,t} = 1$, so we always stop whenever we reach the root. Due to the effectively linear structure of the relevant tree nodes, the stopping probability parameters w can be updated in closed form, as shown in ([38], Theorem 1) via Bayes' theorem as follows:

$$w_{i,t+1} = \frac{p_t(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, c_t = i) w_{i,t}}{p_t(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, c_t \in \{i\} \cup \mathfrak{D}_t(i))}. \tag{5.2.1}$$

Since there is a different tree for each action, $c_t = i$ uniquely identifies a tree, the action does not need to enter in the conditional expressions above. Finally, it is easy to see, by marginalisation and the definition of the stopping probabilities, that the denominator in the above equation can be calculated recursively:

$$p_t(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, c_t \in \{i\} \cup \mathfrak{D}_t(i)) = w_{i,t} p_t(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, c_t = i) + (1 - w_{i,t}) p_t(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, c_t \in \mathfrak{D}_t(i)).$$

Consequently, inference can be performed with a simple forward-backward sweep through a single tree path. In the forward stage, we compute the probabilities of the denominator, until we reach the point where we have to insert a new node. Whenever a new node is inserted in the tree, its weight parameter is initialised to $2^{-d(i)}$. We then go backwards to the root node, updating the weight parameters and the posterior of each model. The only remaining question is how to calculate the individual predictive marginal distributions for each context i in the forward sweep and how to calculate their posterior in the backward sweep. In this work, we associate a linear Bayesian model with each context, which provides this distribution.

5.2.3 The Linear Bayesian Model

In our model we assume that, given $c_t = i$, the next state s_{t+1} is given by a linear transformation of the current state and additive noise $\varepsilon_{i,t}$:

$$s_{t+1} = A_i x_t + \varepsilon_{i,t}, \qquad x_t \triangleq \begin{pmatrix} s_t \\ 1 \end{pmatrix}, \tag{5.2.2}$$

where x_t is the current state vector augmented by a unit basis.[4] In particular, each context models the dynamics via a Bayesian multivariate linear-Gaussian model. For the i-th context, there is a different (unknown) parameter pair (A_i, V_i) where A_i is the *design* matrix and V_i is the *covariance* matrix. Then the next state distribution is:

$$s_{t+1} \mid x_t = x, c_t = i \sim \mathcal{N}(A_i x, V_i).$$

Thus, the parameters ϑ_t which are abstractly shown in Fig. 5.1 correspond to the two matrices A, V. We now define the conditional distribution of these matrices given $c_t = i$.

We can model our uncertainty about these parameters with an appropriate prior distribution p_0. In fact, a conjugate prior exists in the form of the *matrix inverse-Wishart normal* distribution. In particular, given $V_i = V$, the distribution for A_i is matrix-normal, while the marginal distribution of V_i is inverse-Wishart:

$$A_i \mid V_i = V \sim \mathcal{N}(A_i \mid \underbrace{M, C}_{\text{prior parameters}}, V) \tag{5.2.3}$$

$$V_i \sim \mathcal{W}(V_i \mid \overbrace{W, n}). \tag{5.2.4}$$

Here $\mathcal{N}$ is the prior on design matrices, which has a matrix-normal distribution, conditional on the covariance and two prior parameters: M, which is the prior mean and C which is the prior covariance of the dependent variable (i.e., the output). Finally, $\mathcal{W}$ is the marginal prior on covariance matrices, which has an inverse-Wishart distribution with W and n. More precisely, the distributions have the following forms:

$$\mathcal{N}(A_i \mid M, C, V) \propto e^{-\frac{1}{2}\operatorname{tr}\left[(A_i - M)^\top V^{-1}(A_i - M)C\right]}$$
$$\mathcal{W}(V \mid W, n) \propto |V^{-1}W/2|^{n/2} e^{-\frac{1}{2}\operatorname{tr}(V^{-1}W)}.$$

Essentially, the model extends the classic Bayesian linear regression model (e.g., [33]) to the multivariate case via vectorisation of the mean matrix. Since the prior is conjugate, it is relatively simple to calculate the posterior after each observation. For simplicity, and to limit the total number of prior parameters we have to select, we use the same prior parameters (M_i, C_i, W_i, n_i) for all contexts in the tree.

To integrate this with inference in the tree, we must define the marginal distribution used in the nominator of Eq. 5.2.1. This is a multivariate Student-t distribution, so if the posterior parameters for context i at time t are $(M_i^t, C_i^t, W_i^t, n_i^t)$, then this is:

$$p_t(s_{t+1} \mid x_t = x, c_t = i) = \mathit{Student}(M_i^t, W_i^t / z_i^t, 1 + n_i^t),$$

where $z_i^t = 1 - x^\top (C_i^t + xx^\top)^{-1} x$.

[4] While other transformations of s_t are possible, we do not consider them in this work.

Regression Illustration

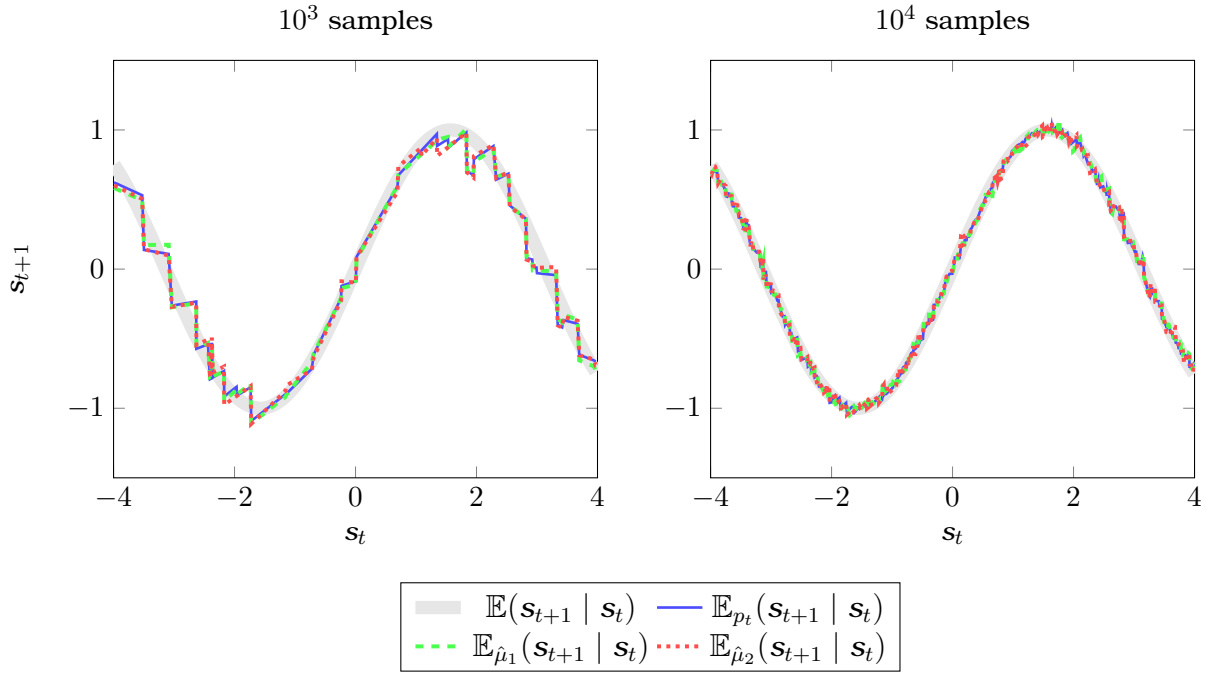


Figure 5.2: Regression illustration. We plot the expected value for the real distribution, the marginal, as well as two sampled models $\hat{\mu}_1, \hat{\mu}_2 \sim p_t(\mu)$.

An illustration of inference using the generalised context tree is given in Fig. 5.2, where the piecewise-linear structure is evident. The s_t variates are drawn uniformly in the displayed interval, while $s_{t+1} \mid s_t = s \sim \mathcal{N}(\sin(s), 0.1)$, i.e., drawn a normal distribution with mean $\sin(s_t)$ and variance 0.1. The plot shows the marginal expectation $\mathbb{E}_{p_t}$, as well as the expectation from two different models sampled from the posterior $p_t(\mu)$.

5.2.4 Approximating the Optimal Policy with Thompson Sampling

Many algorithms exist for finding the optimal policy for a specific MDP μ, or for calculating the expected utility of a given policy for that MDP. Consequently, a simple idea is to draw MDP samples μ_i from the current posterior distribution and then calculate the expected utility of each. This can be used to obtain approximate lower and upper bounds on the Bayes-optimal expected utility by maximising over the set of memoryless policies Π_1. Taking K samples, allows us to calculate the upper and lower bounds with accuracy $O(1/\sqrt{K})$.

$$\max_{\pi \in \Pi_1} \mathbb{E}_p^\pi U \approx \max_{\pi \in \Pi_1} \frac{1}{K} \sum_{i=1}^{K} \mathbb{E}_{\mu_i}^\pi U \leq \frac{1}{K} \sum_{i=1}^{K} \max_{\pi \in \Pi_1} \mathbb{E}_{\mu_i}^\pi U, \qquad \mu_i \sim p_t(\mu). \tag{5.2.5}$$

We consider only the special case $K = 1$, i.e., when we only sample a single MDP. Then the two values are identical and we recover Thompson sampling. The main problems

we have to solve now is how to sample a model and how to calculate a policy for the sampled model.

Sampling a Model from the Posterior

Each model μ sampled from the posterior corresponds to a particular choice of tree parameters. Sampling is done in two steps. The first generates a partition from the tree distribution and the second step generates a linear model for each context in the partition.

The first step is straightforward. We only need to sample a set of weights $\hat{w}_i \in \{0, 1\}$ such that $\mathbb{P}(\hat{w}_i = 1) = w_{i,t}$, as shown in [38] (Rem. 2). This creates a *partition*, with one Bayesian multivariate linear model responsible for each context in the partition.

The second step is to sample a design and covariance matrix pair $(\hat{A}_i, \hat{V}_i)$ for each context i in the partition. This avoids sampling matrices for contexts not part of the sampled tree. As the model suggests, we can first sample the noise covariance by plugging the posterior parameters in Eq. 5.2.4 to obtain $\hat{V}_i$. Sampling from this distribution can be done efficiently using the algorithm suggested by [115]. We then plug in $\hat{V}_i$ into the conditional design matrix posterior Eq. 5.2.3 to obtain a design matrix $\hat{A}_i$ by sampling from the resulting matrix-normal distribution.

The final MDP sample μ from the posterior has two elements. Firstly, a set of contexts $\hat{C}^\mu \subset \bigcup_{a \in \mathcal{A}} \hat{\mathcal{T}}_a$, from all action trees. This set is a partition with associated mapping $f^\mu : \mathcal{S} \times \mathcal{A} \to \hat{C}^\mu$. Secondly, a set of associated design and covariance matrices $\left\{ (A_i^\mu, V_i^\mu) \;\middle|\; i \in \hat{C}^\mu \right\}$ for each context. Then the prediction of the sampled MDP is:

$$\mathbb{P}_\mu(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, a_t) = \mathcal{N}(A^\mu_{f(\boldsymbol{s}_t, a_t)} \boldsymbol{x}_t, V^\mu_{f(\boldsymbol{s}_t, a_t)}), \tag{5.2.6}$$

where $\boldsymbol{x}_t$ is given in Eq. 5.2.2.

Finding a Policy for a Sample via ADP

In order to calculate an optimal policy $\pi^*(\mu)$ for μ, we generate a large number of trajectories from μ using a uniform policy. After selecting an appropriate set of basis functions, we then employ a variant of the least-squares policy iteration (LSPI, [75] algorithm, using least-squares temporal differences (LSTD, [21] rather than LSTDQ. This is possible because since we have μ available, we have access to (5.2.6) and it makes LSPI slightly more efficient.

More precisely, consider the *value function* $V^\pi_\mu : \mathcal{S} \to \mathbb{R}$, defined as:

$$V^\pi_\mu(\boldsymbol{s}) \triangleq \mathbb{E}^\pi_\mu \left(U \mid \boldsymbol{s}_t = \boldsymbol{s} \right).$$

Unfortunately, for continuous $\mathcal{S}$ finding an optimal policy requires approximations. A common approach is to make use of the fact that:

$$V^\pi_\mu(\boldsymbol{s}) = \rho(\boldsymbol{s}) + \gamma \int_{\mathcal{S}} V^\pi_\mu(\boldsymbol{s}') \, \mathrm{d}P^\pi_\mu(\boldsymbol{s}' \mid \boldsymbol{s}),$$

where we assume for simplicity that $\rho(s)$ is the reward obtained at state s. The conditional measure P_μ^π is the transition kernel on $\mathcal{S}$ induced by μ, π, introduced in Section 5.1. We then select a parametric family $v_\omega : \mathcal{S} \to \mathbb{R}$ with parameter $\omega \in \Omega$ and minimise:

$$h(\omega) + \int_{\mathcal{S}} \left\| v_\omega(s) - \rho(s) - \gamma \int_{\mathcal{S}} v_\omega(s') \, \mathrm{d}\hat{P}_\mu^\pi(s'|s) \right\| \mathrm{d}\chi(s), \tag{5.2.7}$$

where h is a regularisation term, χ is an appropriate measure on $\mathcal{S}$ and $\hat{P}_\mu^\pi$ is an empirical estimate of the transition kernel, used to approximate the respective integral that uses P_μ^π. As we can take an arbitrary number of trajectories from μ, π, this can be as accurate as our computational capacity allows.

In practice, we minimise Eq.5.2.7 with a generalised linear model (defined on an appropriate basis) for v_ω while χ need only be positive on a set of representative states. Specifically, we employ a variant of the least-squares policy iteration (LSPI, [75]) algorithm, using the least-squares temporal differences (LSTD, [21]) for the minimisation of Eq. 5.2.7. Then the norm is the euclidean norm and the regularisation term is $h(\omega) = \lambda\|\omega\|$. In order to estimate the inner integral, we take $K_\mathrm{L} \geq 1$ samples from the model so that

$$\begin{aligned} \hat{P}_\mu^\pi(s' \mid s) &\triangleq \frac{1}{K_\mathrm{L}} \sum_{i=1}^{K_\mathrm{L}} \mathbb{I}\left\{ s_{t+1}^i = s' \mid s_t^i = s \right\}, \\ s_{t+1}^i \mid s_t^i = s &\sim P_\mu^\pi(\cdot \mid s), \end{aligned} \tag{5.2.8}$$

where $\mathbb{I}\{\cdot\}$ is an indicator function and P_μ^π is decomposable in known terms. Equation Eq. 5.2.8 is also used for action selection in order to calculate an approximate expected utility $q_\omega(s, a)$ for each state-action pair (s, a):

$$q_\omega(s, a) \triangleq \rho(s) + \gamma \int_{\mathcal{S}} v_\omega(s') \, \mathrm{d}\hat{P}_\mu^\pi(s'|s)$$

Effectively, this approximates the integral via sampling. This may add a small amount[5] of additional stochasticity to action selection, which can be reduced[6] by increasing K_L.

Finally, we optimise the policy by approximate policy iteration. At the j-th iteration we obtain an improved policy $\hat{\pi}_j(a \mid s) \propto \mathbb{P}[a \in \arg\max_{a' \in \mathcal{A}} q_{\omega_{j-1}}(s, a')]$ from ω_{j-1} and then estimate ω_j for the new policy.

5.2.5 Complexity

We now analyse the computational complexity of our approach, including the online complexity of inference and decision making, and of the sampling and ADP taking place every episode. It is worthwhile to note two facts. Firstly, that the complexity

[5] Generally, this error is bounded by $O(K_\mathrm{L}^{-1/2})$.

[6] We remind the reader that Thompson sampling itself results in considerable exploration by sampling an MDP from the posterior. Thus, additional randomness may be detrimental.

bounds related to the cover tree depend on a constant c, which however depends on the distribution of samples in the state space. In the worst case (i.e., a uniform distribution), this is bounded exponentially in the dimensionality of the actual state space. While we do not expect this to be the case in practice, it is easy to construct a counterexample where this is the case. Secondly, that the complexity of the ADP step is largely independent of the model used, and mostly depends on the number of trajectories we take in the sampled model and the dimensionality of the feature space.

First, we examine the total computation time that is required to construct the tree.

Corollary 5.1. *Cover tree construction from t observations takes $O(t \ln t)$ operations.*

Proof. In the cover tree, node insertion and query are $O(\ln t)$ ([16], Theorems 5, 6). Then note that $\sum_{k=1}^{t} \ln k \leq \sum_{k=1}^{t} \ln t = t \ln t$. ∎

At every step of the process, we must update our posterior parameters. Fortunately, this also takes logarithmic time as we only need to perform calculations for a single path from the root to a leaf node.

Lemma 5.1. *If $\mathcal{S} \subset \mathbb{R}^m$, then inference at time step t has complexity $O(m^3 \ln t)$.*

Proof. At every step, we must perform inference on a number of nodes equal to the length of the path containing the current observation. This is bounded by the depth of the tree, which is in turn bounded by $O(\ln t)$ from [16] (Lemma 4.3). Calculating Eq. 5.2.1 is linear in the depth. For each node, however, we must update the linear-Bayesian model, and calculate the marginal distribution. Each requires inverting an $m \times m$ matrix, which has complexity $O(m^3)$. ∎

Finally, at every step we must choose an action through value function look-up. This again takes logarithmic time, but there is a scaling depending on the complexity of the value function representation.

Lemma 5.2. *If the LSTD basis has dimensionality m_L, then taking a decision at time t has complexity $O(K_L m_L \ln t)$.*

Proof. To take a decision we merely need to search in each action tree to find a corresponding path. This takes $O(\ln t)$ time for each tree. After Thompson sampling, there will only be one linear model for each action tree. LSTD takes K_{L} operations, and requires the inner product of two m_L-dimensional vectors. ∎

The above lemmas give the following result:

Theorem 5.1. *At time t, the online complexity of CTBRL is $O((m^3 + K_L m_L) \ln t)$.*

We now examine the complexity of finding a policy. Although this is the most computationally demanding part, its complexity is not dependent on the cover tree structure or the probabilistic inference method used. However, we include it here for completeness.

Lemma 5.3. *Thompson sampling at time t is $O(tm^3)$.*

Proof. In the worst case, our sampled tree will contain all the leaf nodes of the reduced tree, which are $O(t)$. For each sampled node, the most complex operation is Wishart generation, which is $O(m^3)$ [115]. ∎

Lemma 5.4. *If we use n_s samples for LSTD estimation and the basis dimensionality is m_L, this step has complexity $O(m_L^3 + n_s(m_L^2 + K_L m_L \ln t))$.*

Proof. For each sample we must take a decision according to the last policy, which requires $O(K_{\mathrm{L}} m_L \ln t)$ as shown previously. We also need to update two matrices (see [20]), which is $O(m_L^2)$. So, $O(n_s(m_L^2 + K_{\mathrm{L}} m_L \ln t))$ computations must be performed for the total number of the selected samples. Since LSTD requires an $m_L \times m_L$ matrix inversion, with complexity $O(m_L^3)$, we obtain the final result. ∎

From Lemmas 5.2 and 5.4 it follows that:

Theorem 5.2. *If we employ API with K_A iterations, the total complexity of calculating a new policy is $O(tm^3 + K_A(m_L^3 + n_s(m_L^2 + K_L m_L \ln t)))$.*

Thus, while the online complexity of CTBRL is only logarithmic in t, there is a substantial cost when calculating a new policy. This is only partially due to the complexity of sampling a model, which is manageable when the state space has small dimensionality. Most of the computational effort is taken by the API procedure, at least as long as $t < (m_L/m)^3$. However, we think this is unavoidable no matter what the model used is.

The complexity of Gaussian process (GP) models is substantially higher. In the simplest model, where each output dimension is modelled independently, inference is $O(mt^3)$, while the fully multivariate tree model has complexity $O(m^3 t \ln t)$. Since there is no closed form method for sampling a function from the process, one must resort to iterative sampling of points. For n points, the cost is approximately $O(nmt^3)$, which makes sampling long trajectories prohibitive. For that reason, in our experiments we only use the mean of the GP.

5.3 Empirical Evaluation

We conducted two sets of experiments to analyse the offline and the online performance. We compared CTBRL with the well-known LSPI algorithm [75] for the offline case, as well as an online variant [27] for the online case. We also compared CTBRL with Linear Bayesian Reinforcement Learning (LBRL, [137]) and finally GP-RL, where we simply replaced the tree model with a Gaussian process. For CTBRL and LBRL we use Thompson sampling. However, since Thompson sampling cannot be performed on GP models, we use the mean GP instead. In order to compute policies given a model, all model-based methods use the variant of LSPI explained in Section 5.2.4. Hence, the

only significant difference between each approach is the model used, and whether or not they employ Thompson sampling.

A significant limitation of Gaussian processes is that their computational complexity becomes prohibitive as the number of samples becomes extremely large. In order to make the GP model computationally practical, the greedy approximation approach introduced by [45] has been adopted. This is a kernel sparsification methodology which incrementally constructs a dictionary of the most representative states. More specifically, an *approximate linear dependence* analysis is performed in order to examine whether a state can be approximated sufficiently as a linear combination of current dictionary members or not.

We used one preliminary run and guidance from the literature to make an initial selection of possible hyper-parameters, such as the number of samples and the features used for LSTD and LSTD-Q. We subsequently used 10 runs to select a single hyper-parameter combination for each algorithm-domain pair. The final evaluation was done over an independent set of 100 runs.

For CTBRL and the GP model, we had the liberty to draw an arbitrary number of trajectories for the value function estimation. We drew 1-step transitions from a set of 3000 uniformly drawn states from the *sampled* model (the mean model in the GP case). We used 25 API iterations on this data.

For the offline performance evaluation, we first drew rollouts from $k = \{10, 20, \ldots, 50, 100, \ldots, 1000\}$ states drawn from the *true environment's* starting distribution, using a uniformly random policy. The maximum horizon of each rollout was set equal to 40. The collected data was then fed to each algorithm in order to produce a policy. This policy was evaluated over 1000 rollouts on the environment.

In the online case, we simply use the last policy calculated by each algorithm at the end of the last episode, so there is no separate learning and evaluation phase. This means that efficient exploration must be performed. For CTBRL, this is done using Thompson sampling. For online-LSPI, we followed the approach of [27], who adopts an ϵ-greedy exploration scheme with an exponentially decaying schedule $\epsilon_t = \epsilon_d^t$, with $\epsilon_0 = 1$. In preliminary experiments, we found $\epsilon_d = 0.997$ to be a reasonable compromise. We compared the algorithms online for 1000 episodes.

5.3.1 Domains

We consider two well-known continuous state, discrete-action, episodic domains. The first is the *mountain car* domain and the second is the *inverted pendulum* domain.

Mountain Car

The aim in this domain is to drive an underpowered car to the top of a hill. Two continuous variables characterise the vehicle state in the domain, its position and its velocity. The objective is to drive an underpowered vehicle up a steep valley from a

randomly selected position to the right hilltop (at position > 0.5) within 1000 steps. There are three actions: forward, reverse and zero throttle. The received reward is -1 except in the case where the target is reached (zero reward). At the beginning of each rollout, the vehicle is positioned to a new state, with the position and the velocity uniformly randomly selected. The discount factor is set to $\gamma = 0.999$. An equidistant 4×4 grid of RBFs over the state space plus a constant term is selected for LSTD and LSPI.

Inverted Pendulum

The goal in this domain, is to balance a pendulum by applying forces of a fixed magnitude (50 Newtons). The state space consists of two continuous variables, the vertical angle and the angular velocity of the pendulum. There are three actions: no force, left force or right force. A zero reward is received at each time step except in the case where the pendulum falls. In this case, a negative (-1) reward is given and a new episode begins. An episode also ends with 0 reward after 3000 steps, after which we consider that the pendulum is successfully balanced. Each episode starts by setting the pendulum in a perturbed state close to the equilibrium point. More information about the specific dynamics can be found at [75]. The discount factor is set to $\gamma = 0.95$. The basis we used for LSTD/LSPI, was equidistant 3×3 grid of RBFs over the state space following the suggestions of [75]. This was replicated for each action for the LSTD-Q algorithm used in LSPI.

5.3.2 Results

In our results, we show the average performance in terms of number of steps of each method, averaged over 100 runs. For each average, we also plot the 95% confidence interval for the accuracy of the mean estimate with error bars. In addition, we show the 90% percentile region of the runs, in order to indicate inter-run variability in performance.

Figure 5.3(a) shows the results of the experiments in the *offline* case. For the *mountain car*, it is clear that CTBRL is significantly more stable compared to GPRL and LSPI. In contrast to the other two approaches, CTBRL needs only a small number of rollouts in order to discover the optimal policy. For the *pendulum* domain, the performance of both CTBRL and GPRL is almost perfect, as they need only about twenty rollouts in order to discover the optimal policy. On the other hand, LSPI despite the fact that manages to find the optimal policy frequently, around 5% of its runs fail.

Figure 5.3(b) shows the results of the experiments in the *online* case. For the *mountain car*, CTBRL managed to find an excellent policy in the vast majority of runs, while converging earlier than GPRL and LSPI. Moreover, CTBRL presents a more stable behaviour in contrast to the other two. In the *pendulum* domain, the performance difference relative to LSPI is even more impressive. It becomes apparent that both

CTBRL and GPRL reach near optimal performances with an order of magnitude fewer episodes than LSPI, while the latter remains unstable. In this experiment, we see that CTBRL reaches an optimal policy slightly before GPRL. Although the difference is small, it is very consistent.

The success of CTBRL over the other approaches can be attributed to a number of reasons. Firstly, it could be a better model. Indeed, in the offline results for the mountain car domain, where the starting state distribution is uniform, and all methods have the same data, we can see that CTBRL has a far better performance than everything else. The second could be the more efficient exploration afforded by Thompson sampling. Indeed, in the mountain car online experiments we see that the LBRL performs quite well (Fig. 5.3(b)), even though its offline performance is not very good (Fig. 5.3(a)). However, Thompson sampling is not sufficient for obtaining a good performance, as seen by both the offline results and the performance in the pendulum domain.

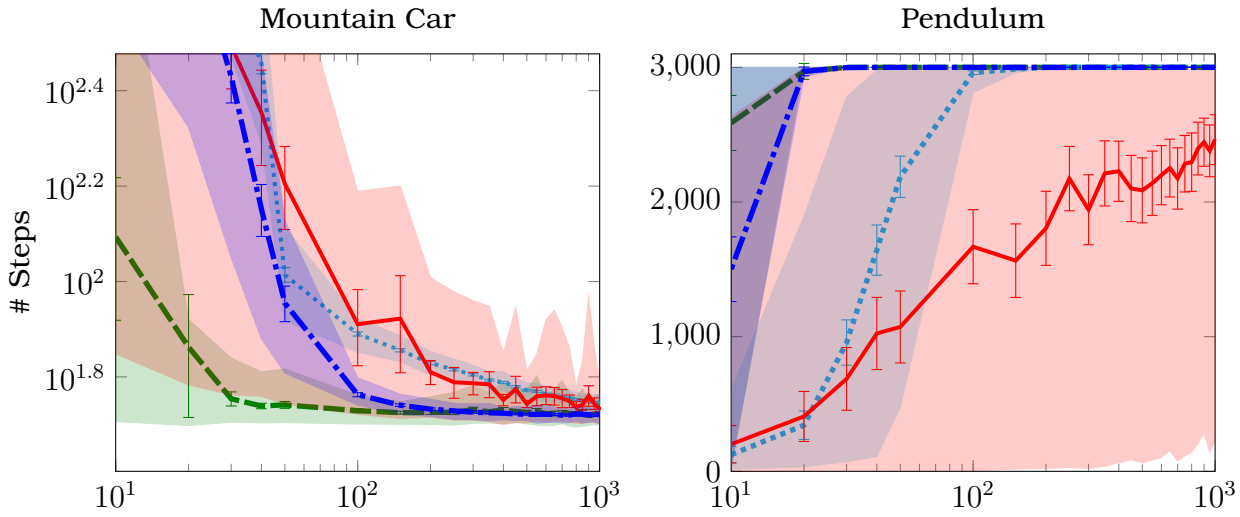
5.4 Summary

We proposed a computationally efficient, fully Bayesian approach for the exact inference of unknown dynamics in continuous state spaces. The total computation for inference after t steps is $O(t \ln t)$, in stark contrast to other non-parametric models such as Gaussian processes, which scale $O(t^3)$. In addition, inference is naturally performed online, with the computational cost at time t being $O(\ln t)$.

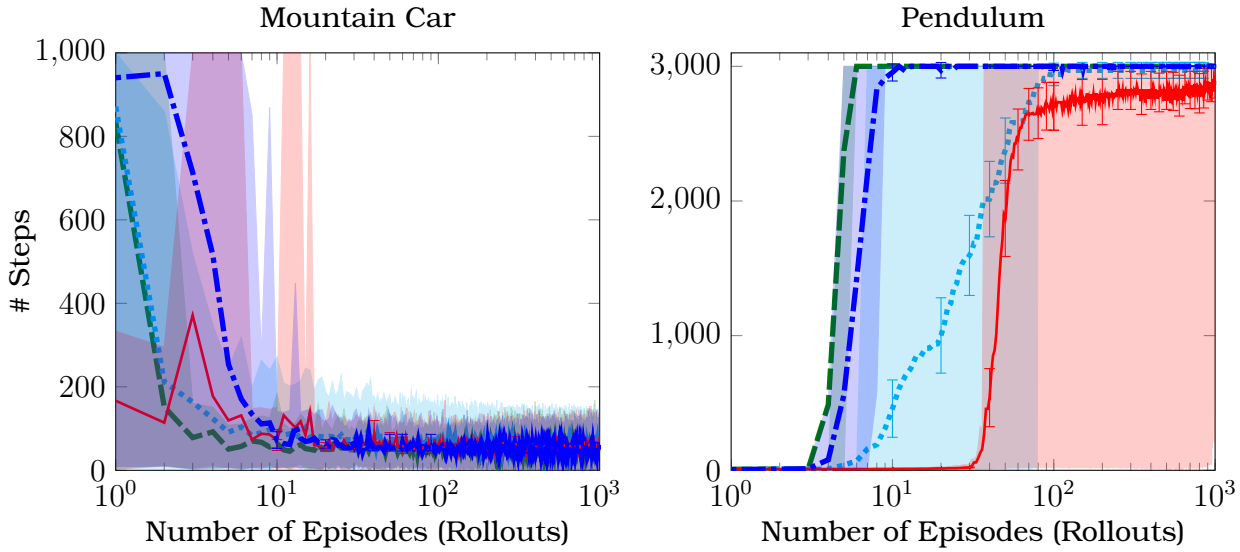
In practice, the computational complexity is orders of magnitude lower for cover trees than GP, even for these problems. We had to use a dictionary and a lot of tuning to make GP methods work, while cover trees worked out of the box. Another disadvantage of GP methods is that it is infeasible to implement Thompson sampling with them. This is because it is not possible to directly sample a function from the GP posterior. Although Thompson sampling confers no advantage in the offline experiments (as the data there were the same for all methods), we still see that the performance of CTBRL is significantly better on average and that it is much more stable.

Experimentally, we showed that cover trees are more efficient both in terms of computation and in terms of reward, relative to GP models that used the same ADP method to optimise the policy and to a linear Bayesian model which used both the same ADP method and the same exploration strategy. We can see that overall the linear model performs significantly worse than both GP-RL and CTBRL, though better than ϵ-greedy LSPI. This shows that the main reason for the success of CTBRL is the cover tree inference and not the linear model itself, or Thompson sampling.

CTBRL is particularly good in online settings, where the exact inference, combined with the efficient exploration provided by Thompson sampling give it an additional advantage. We thus believe that CTBRL is a method that is well-suited for exploration in unknown continuous state problems. Unfortunately, it is not possible to implement Thompson sampling in practice using GPs, as there is no reasonable way to sample a



(a) Offline results



(b) Online results

Figure 5.3: Experimental evaluation. The dashed line shows CTBRL, the dotted line shows LBRL, the solid line shows LSPI, while the dash-dotted line shows GPRL. The error bars denote 95% confidence intervals for the mean (i.e., statistical significance). The shaded regions denote 90% percentile performance (i.e., robustness) across runs. In all cases, CTBRL converges significantly quicker than the other approaches. In addition, as the percentile regions show, it is also much more stable than LBRL, GPRL and LSPI.

function from the GP posterior. Nevertheless, we found that in both online and offline experiments (where Thompson sampling should be at a disadvantage) the cover tree method achieved superior performance to Gaussian processes.

Chapter 6

A Reinforcement Learning Agent for Ms. PacMan

Digital games have received enormous research interest in Artificial Intelligence (AI) community, during the last decades. As game environments become more complex and realistic through the years, they offer a range of fascinating toy examples that capture the complexity of real-world situations while maintaining the controllability of computer simulations. A key problem is the development of AI driven agents that will be competitive with human intelligence as well as adaptive to dynamically changed environments [53]. These agents can take a variety of roles such as player's opponents, teammates or other non-player characters.

Reinforcement learning (RL) covers the capability of learning from experience [66, 118] gained by interacting within an unknown environment. Thus, offers a very attractive and powerful platform for learning to control an agent in unknown environments with limited prior knowledge. At the same time, games are ideal testbed environments for the RL paradigm, since they are goal-oriented sequential decision problems, where each decision can have long-term effect. Games also exhibit a number of interesting properties that are important in RL research. For example, a game environment can be either static or dynamic, there can be either single-agent or two-player or multi-agent problems, transitions can be either deterministic or stochastic, and game environments can be either fully known or partially observable.

A number of strategies based on the reinforcement learning scheme has been proposed in a variety of classical games, such as *Chess*, *Backgammon*, *Computer Go*, *Tetris*, etc. (see [122] for an overview). Among them, the arcade video game Ms. Pac-Man constitutes a really challenging domain. That makes Ms. Pac-Man really attractive is its simplicity of playing in combination with the complex strategies that are required to obtain a good performance [123]. The game of Ms. Pac-Man meets all the criteria of a reinforcement learning task. The environment is difficult to be predicted, as the ghost's behaviour is stochastic and their tracks are unpredictable. The reward function can be easily defined covering particular game events and score requirements. Furthermore, there is a small action space consisting of the four directions in which Ms. Pac-Man can move (up, down, right, left) at each time step. However, a difficulty is encountered with designing an appropriate state space encoding for the particular domain. Specifically, incorporating a large number of features for describing a single game snapshot, may limit the efficiency of the agent as the complexity of the problem grows exponentially with the number of variables. On the other hand, a poor representation does not allow the agent to distinguish the majority of different game situations. Therefore, establishing an expressive state representation is of central interest, as allows the development of an efficient agent which will be able to discover policies in reasonable time.

In this chapter, our study is especially focused on designing an appropriate state space representation for building an efficient RL agent to the Ms. Pac-Man game. The proposed state representation is informative as incorporates all the necessary knowledge about any game snapshot. At the same time, it presents an abstract description which achieves to reduce the computational cost and accelerate the learning procedure without compromising the decision quality. It has been demonstrated that providing a proper feature set as input to the learner is of outmost importance for simple reinforcement learning algorithms. The last observation constitutes the main contribution of our study and suggests the need of a careful modeling of the domain. The on-policy reinforcement learning algorithm, SARSA(λ), has been used for decision making. Several experiments have been conducted where the learning capabilities of the proposed methodology are measured along with its efficiency in discovering optimal policies in unknown mazes. It is worth mentioning that a direct comparison is not possible as different versions of the game (i.e. Ms. PacMan simulators) have been employed in the literature. Nevertheless, we believe that the proposed agent yields very promising results along with a considerable improved performance.

The remainder of this chapter is organised as follows. In Section 6.1, the environment and rules of the Pac-Man game are briefly described. Section 6.2 reviews reinforcement learning methodologies in games. The proposed RL-PacMan agent is presented in Section 6.3, consisting of the adopted state representation along with the general temporal-difference (TD) scheme used to solve the prediction problem. An experimental illustration is given in Section 6.4, followed by concluding remarks in Section 6.5

6.1 The Game of Ms. PacMan

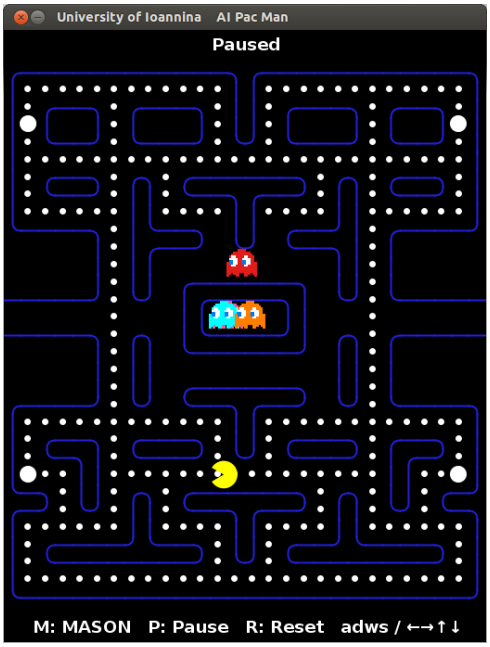


Figure 6.1: A screenshot of the Pac-Man game in a typical maze (*Pink maze*)

Pac-Man is a real-time arcade video-game released in 1980 and since then has reached immense success as it is considered to be one of the most popular video games up to date. The player guides the main character called Pac-Man through a number of mazes (Figure 6.1 illustrates a typical such maze) filled with pills. The game's objective, is the collection of all these pills, avoiding or chasing the ghosts (*enemies*) which roam the maze trying to capture the Pac-Man. There are four different ghosts: Blinky (red), Pinky (pink), Inky (cyan) and Sue (yellow). At the start of a game, Pac-Man has at his disposal three lives and a life is taken away every time he collides with a non scared ghost. After losing a life, the ghosts as well as Pac-Man return to their initial positions on that maze. A level is cleared when all the pills are consumed by the Pac-Man. The game ends naturally when Pan-Man has spent all lives.

Four special pills are located near the corners of each maze, called *power pills*. Whenever a power pill is consumed by the Pac-Man the ghosts become edible, i.e. the ghosts turn blue for a short time period, and they are not constitute threat for the Pac-Man. As long as the ghosts are in that state, their movement speed is decreased and they are forced to reverse their direction in an attempt to move away from the Pac-Man. Each time a ghost is eaten, it returns to the lair at the center of the maze and remains there for a short time period before joining the maze again. On the opposite side, Pac-Man is able to eat them and gains $200 * n$ points for each ghost eaten, where $n \in \{1, 2, 3, 4\}$ is the total number of ghosts eaten after taken the last power pill. Moreover, the Pac-Man gains 10 points for each pill eaten, and 50 points per power pill. As it becomes apparent, the player would want to eat all four ghosts per power pill in

order to gain the highest possible score. For example, the maximum achievable score for the maze represented at Fig. 6.1 is: $220 \times 10 + 4 \times 50 + 4 \times (200 + 400 + 800 + 1600) = 14400$[1].

In the original version of Pac-Man, ghosts move on a complex but deterministic route, so it is possible for the player to discover easy a deterministic sequence of actions (pattern) without the need of any observation. On the other side, in Ms. Pac-Man, randomness is added to the movements of each ghost. Therefore there is not a single optimal policy and observations are necessary for an optimal decision making at each time step. In our case, ghosts moved randomly in 20% of the time and straight towards Ms. Pac-Man in the remaining 80%. In both cases, ghosts are allowed to make a move only if they are at a junction in the maze. In contrast, they can only traverse forward if they are on a corridor (a corridor lies between two junctions).

6.2 Reinforcement Learning in Games

Games constitute a really challenging area for reinforcement learning research, with a large number of applications. A number of popular techniques such as Temporal-difference learning [119], Monte-Carlo tree search [70], evolutionary reinforcement learning [86], etc., have been applied with success until now. In the most of the cases, the proposed RL approaches are competitive with the other AI techniques as well as human experts. Nevertheless, a lot of challenges and open problems left to solve. For example, the appropriate choice of algorithm is just one among many factors that could drive to success or failure, while in most of cases is not even the most significant factor. At the same time, a number of components such as the appropriate choice of state representation, the encoding of domain knowledge as well as the proper setting of parameters can have great influence at the efficiency of any application of RL.

Backgammon is the game where reinforcement learning reached its first great success, achieving to reach and exceed the level of world's strongest grandmasters. Tesauro's TD-Gammon [125, 126] is a combination of the TD(λ) learning algorithm and nonlinear function approximation. More specifically, a multilayer neural network consisted of a single hidden layer is used for estimating the value of any board position. The input of the network is a representation of the board position and its output is an estimation of the probability of winning starting from that state. To accomplish this, rewards is defined as zero for all time steps except those on which the game is won ($+1$ for winning). After its dice roll, the move that will lead to the position with the highest possible value, will be selected. Then, the TD error is back-propagated in order to adjust the network's weights. As a huge number of games is required for the training of the network, TD-Gammon overcomes this handicap by playing against itself and learning from the outcomes. In that way, TD-Gammon is considered as a self-teaching methodology which results in a program with incredible playing abilities.

[1] In the original version of the game, extra fruits at random time intervals and prizes appear and roam around the maze.

Chess is a two-player strategy, full informative, zero-sum board game. Its popularity, prestige and complexity has made it one of the most actively AI research games. In spite of the research effort in the last decades, *Chess* is one of the game where the playing level even of the best RL approaches is still modest, compared to the great success in many other games. There are a number of reasons for the weakness of RL to be competitive. First of all, the extraction of sufficient features for the encoding of the game structure is not trivial. In addition to that, *Chess* is also hard for state-space sampling approaches (e.g. UCT [70]), even when augmented with heuristic evaluation functions. Another factor that make it hard for the RL is the deterministic nature of the *Chess*. In that way, agents have to explore actively the state space in order to gain experience. A common approach for playing chess is the combination of the evaluation functions with a multi-step lookahead algorithm (e.g. minimax search). In [10], the idea of TD-Leaf is presented for the training of an evaluation function. On the other hand, a modification of the TD-Leaf algorithm, called TreeStrap, has been proposed to [145]. It has been shown that TreeStrap achieves to approximately reach the same level as TD-Leaf by using expert trainers. An interesting remark is that for the first time, an algorithm learns to play master level *Chess* entirely through self play.

Similarly to *Chess*, *Go* is a two-player, zero sum board game (standard board size is 19×19) that originated in ancient China. To date, the best Go programs play at the Master level, way below the level of the best human players. Due to its difficulty, smaller and simpler boards such as 9×9 and 13×13 are typically used by many programs. Go has proved to poses a number of challenges because of is intuitive nature, and requires a different approach in contrast to other games. Firstly, the large search space caused by the great number of legal model along with the game's length, are often cited as the principal reasons for the difficulty of the *Go*. Secondly, the discovery of a reasonable and suitable evaluation function has been proved to be a far from easy task. In [113], the reinforcement learning architecture, Dyna-2, has been presented and applied to the Computer Go reached pretty good performance on small boards. The main idea in Dyna-2 is the combination of two separated memories: a permanent memory that is updated from real experience and a transient memory that is updated from simulated experience. Both memories use linear function approximation to form a compact representation of the state space, and are updated by temporal-difference (TD) learning. Due to the lack of good evaluation functions, a number of Monte-Carlo sampling methods for board evaluation have been successfully applied. In that way, UCT and the other MCTS algorithms constitute the basis of all state-of-the-art Go programs.

Tetris is a popular single-player game played on a two-dimensional board (10×20 grid). A number of factors have led Tetris to become one of the most popular game for testing and benchmarking RL algorithms (see [127], for an overview). First of all, Tetris fits the MDP properties quite well as a fully-observable game with randomness. In addition to that, it is known to be computationally hard to solve. One of the first works

that apply RL on Tetris is that proposed in [15]. There the state space is represented as a 21-dimensional feature vector consisting of the individual column heights (10 features), the absolute differences between the heights of adjacent columns (9 features), the maximum height among the columns and the number of holes in the wall. The λ-policy iteration algorithm, which generalizes the standard value iteration and policy iteration algorithms, is applied in order approximate the value function as a linear model using simulations. Two other reinforcement learning works reused the above feature representation. The first one applies a natural policy gradient method [67], whereas the second one applies a linear programming approach [49]. In [76], the least squares policy iteration (LSPI) method with some original features is applied at Tetris. In contrast to the λ-policy iteration where samples are collected at each iteration, the samples are collected only once at the beginning using a hand-crafted policy. An interesting remark is that the LSPI approach shows interesting convergence properties compared to the λ-policy iteration method.

6.2.1 Reinforcement learning in PacMan

In recent years, Pac-Man style games have received great attention in Artificial Intelligence research. The specific works can be divided into two main categories: those that use AI techniques and those which are based on hand-coded approaches. In this section, we focus on works that fall in the former group and briefly discuss the most related ones.

A reinforcement learning approach has been presented in [19], where a neural network has been used for the estimation of the action-value function. A number of features has been extracted and given as inputs to the neural network. It has been also shown that in unknown mazes a single network for all actions outperforms multiple action networks holding better generalization capabilities. In [26], the learning efficiency of temporal difference learning and evolutionary algorithms has been analysed, showing that under the specific experimental configuration, evolution outperforms the temporal difference learning. A different approach to playing Ms Pac-Man has been proposed in [123]. The aim is the automatic construction of a simple rule based policy. Rules are organised into action modules and a decision about which direction to move in is made based on priorities assigned to the modules in the agent. The cross-entropy optimisation algorithm is used for learning policies that play well.

Due to the successful application of MCTS in games and more specifically in the game of *Computer Go*, the interest in developing MCTS agents for Ms. PacMan has increased. In [110], an MCTS agent has been developed using a $\max^n$ tree search and modeling Pac-Man as a 5-player game. A tree node is defined as the target for PacMan in the maze, while MCTS determines the best move discovering the optimal route to the target. An achievement of MCTS on Pac-Man is the one presented in [61], where an agent is designed to solve the problem of avoiding *pincer* moves (every escape path for Pac-Man is blocked) of the ghosts. A number of MCTS agents has also

proposed for achieving specific goals in the Ms. Pac-Man game, such as ghost avoidance [131] and endgame situations [132]. Recently, in [97] a real-time MCTS approach has been proposed for controlling the Pac-Man character. For the enhancement of the performance of MCTS in a real-time environment such as Pac-Man game, the search tree is reused between successive moves where the stored values are decayed by a factor γ. Three different tactics have also been adopted for determining the objective of Pac-Man at each turn.

6.2.2 Challenges in Games

As pointed out previously, a suitable representation is of central interest in applications of reinforcement learning. A rich domain knowledge about game can assist in the representation design. At the same time, extracting relevant information from sensors is the main source for obtaining the representation. On the other side, an efficient feature mapping of the input has the dual advantage of: abstract away less relevant information, and map similar situations to the same feature vectors, promoting generalization. Last but not least, exploration is a central factor in driving the agent towards an optimal solution. In most games, the exploration stems from the game itself, due to the game dynamics (Backgammon) or the agent's opponents. Nevertheless, Boltzmann and ϵ-greedy action selection are the most commonly used form of exploration.

6.3 The Reinforcement Learning PacMan Agent

This section discusses the main components upon which the proposed RL-PacMan agent is built up. First, a description of the proposed state space encoding is provided. In the following, the reinforcement learning scheme used for decision making is presented. Finally, we present an extension of proposed agent, where the MCTS method is used for selecting the most appropriate *target* suggested to be followed by the agent.

6.3.1 The Proposed State Space Representation

The state space representation is of central interest for the development of an efficient agent. A suitable representation plays main role in system modeling as well as in decision making mechanism. At each time step, the agent has to make decisions according to the observations received by the environment. In that way, the state should represent the internal behaviour of system dynamics by modeling a relationship between inputs and outputs. In particular, the description of the state space in the Ms. Pac-Man game should incorporate useful information about his position, the food (pills, edible ghosts) as well as the ghosts. An ideal state space representation for Ms. Pac-Man should include all the raw sensory information, such as:

- the relative position and direction of Ms. Pac-Man in the maze,

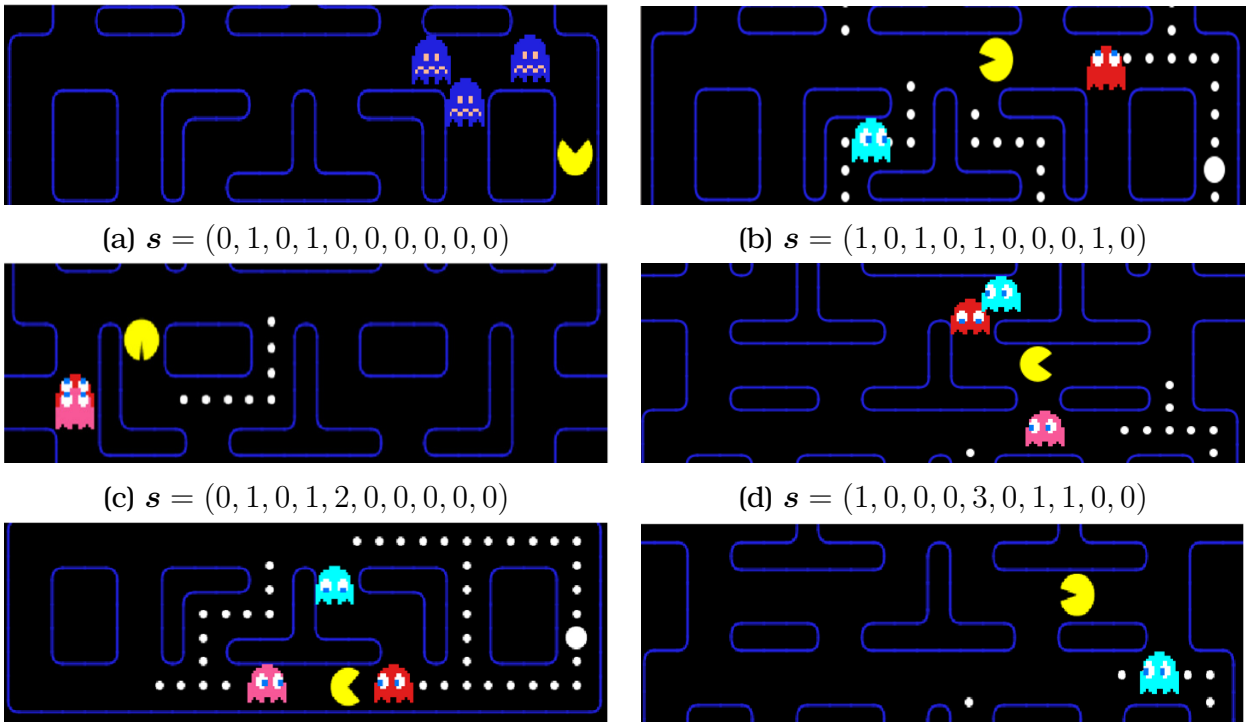
- the relative position and direction of each one of the ghosts,
- the ghosts' state. It declares whether or not a ghost is edible, and if so, for how long remains at the specific situation,
- the existence of a pill or power-pill at a specific position,
- the number of pills/power-pills left,
- the number of lives left.

Although its seemingly benefits, the adoption of such a detailed state space representation can lead to a number of undesirable effects, such as: high computational complexity, low convergence rate, high demands on resources, etc.. Incorporating all the above information, the size of the resulting state space becomes insufficiently large, making the task of discovering a proper policy impractical. In spite of the above mentioned handicaps, a little effort has been paid in the direction of seeking a reasonable and informative state structure.

Considering all the above, our study has focused on the careful construction of an abstract state space description that will be able to incorporate all the useful information. In our approach, the state space is structured as a 10-dimensional, discrete valued, feature vector, $\boldsymbol{s} = (s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10})$. A detailed description of each one of them, is given below:

- The first four (4) features, $(s_1, \ldots, s_4)$, are binary and indicate the existence (1) or not (0) of the wall in the Ms. Pac-Man's four wind directions (north, west, south, east), respectively. Some characteristic examples are illustrated in Fig. 6.2. For example, the state feutures $(s_1 = 0, s_2 = 1, s_3 = 0, s_4 = 1)$ indicates that the Ms. Pac-Man is found in a horizontal corridor (Fig. 6.2(a)). On the other hand, state values $(s_1 = 1, s_2 = 0, s_3 = 1, s_4 = 0)$ means that Ms. Pac-Man is located between a west and east wall (Fig. 6.2(b)). When the Ms. Pac-Man is location at a junction, the first four features are equal to zero.

- The fifth feature, s_5, suggests the direction of the nearest *target* where it is preferable for the Ms. Pac-Man to move. It takes four (4) values, between 0 to 3, that correspond to the four wind directions. The *desired target* depends directly on the distance between the Ms. Pac-Man and the nearest ghost to agent. In particular, when the Ms. Pac-Man is going to be trapped by the ghosts (i.e. at least one ghost with distance less than eight (8) steps is moving threateningly towards Ms. Pac-Man), the direction to the closest *safe* exit (*escape junction*) has been chosen (Fig.6.2(d)). In any other case, the specific feature takes the direction to the closest food, either it is a pill or an edible ghost, with priority to be given to the most valuable among them. Roughly speaking, if an edible ghost exists within a maximum distance of five (5) steps, the direction to the particular ghost will be selected (Fig.6.2(a)). On the other hand, it takes the direction that leads to the



(a) $\boldsymbol{s} = (0, 1, 0, 1, 0, 0, 0, 0, 0, 0)$ (b) $\boldsymbol{s} = (1, 0, 1, 0, 1, 0, 0, 0, 1, 0)$

(c) $\boldsymbol{s} = (0, 1, 0, 1, 2, 0, 0, 0, 0, 0)$ (d) $\boldsymbol{s} = (1, 0, 0, 0, 3, 0, 1, 1, 0, 0)$

(e) $\boldsymbol{s} = (1, 0, 1, 0, 3, 0, 1, 0, 1, 1)$ (f) $\boldsymbol{s} = (1, 0, 1, 0, 1, 0, 0, 0, 0, 0)$

Figure 6.2: Representative game situations along with their state description

nearest dot (Fig.6.2(c,f)). Note here that for calculating the distance as well as the direction between Ms. Pac-Man and *target*, the well-known A* search algorithm [58] has been used for finding the shortest path.

- The next four features, $(s_6, \ldots, s_9)$, are binary and specify the situation of any direction (north, west, south, east) in terms of a *direct* ghost threat. If a ghost with distance less that eight steps (8) is moving towards Ms. Pac-Man from a specific direction, the feature that corresponds in that direction set equal to 1. An example given in Fig.6.2(d), where the Ms. Pac-Man is approached by two ghosts. More specifically, the first ghost approaches the agent from the east ($s_7 = 1$) and the other one from the south direction ($s_8 = 1$).

- The last feature indicates whether the Ms. Pac-Man is trapped in a corridor or not. The Ms. Pac-Man is pointed out as trapped only in the case where no *safe* junction is available (Fig.6.2(e)). In all other cases the Ms. Pac-Man is considered as *free* (Fig.6.2(a, b, c, d, f)). The specific feature is very important as informs the agent whether or not can wander freely inside the maze.

Table 6.3.1 summarizes the proposed state space representation. Obviously, its size is quite small containing only $4 * 2^9 = 2048$ states. Taking also into account that a lot of the above state combinations is inactive (e.g. there is not possible for the agent to be enclosed by walls, $(s_1 = 1, s_2 = 1, s_3 = 1, s_4 = 1)$), the specific number is much less.

This fact allows the construction of a computationally efficient RL agent without the need of any approximation scheme. Last but not least, the adopted reasonable state space combined with the small action space speed up the learning process and enables the agent to discover optimal policies with sufficient generalization capabilities.

Table 6.1: A summary of the proposed state space representation

Feature	***Range***	***Source***
$[s_1\ s_2\ s_3\ s_4]$	$\{0, 1\}$	Ms. Pac-Man view
s_5	$\{0, 1, 2, 3\}$	target direction
$[s_6\ s_7\ s_8\ s_9]$	$\{0, 1\}$	ghost threat direction
s_{10}	$\{0, 1\}$	trapped situation

6.3.2 SARSA Algorithm for Online Learning

The reinforcement learning scheme has been adopted in order to discover a proper policy, π, based on which our agent selects the most appropriate action at each state of the game. SARSA [118] is one of the most popular *on-policy* reinforcement learning algorithm that estimates the policy being followed. As it belongs to the family of temporal difference control methods, it is naturally implemented in an online, fully incremental way without the need of waiting until the end of an episode. In addition to that, SARSA algorithm learns from raw experience without a model of the environment's dynamics. For the above-mentioned reasons, SARSA constitutes a suitable platform for our control problem.

As a model-free control method, SARSA is based on the estimation of the action-value function, Q. Learning a policy therefore means updating the Q-function to make it more accurate. One important aspect of model-free algorithms is that there is a need for exploration. To account for potential inaccuracies in the Q-function, the agent must try out different actions to explore the environment for finding possible better policies. The ϵ-greedy action selection strategy is an effective means of balancing exploration and exploitation in reinforcement learning. According to that, the agent behaves greedily most of the time, but with small probability, ϵ, selects an action uniformly random.

Due to the small state space and the finite number of actions, the value estimations of all state-action pairs can be kept in the main memory being stored in a table (one entry for each state-action pair), known as the tabular case. Thus, the SARSA scheme performs updates to individual Q-value entries in this table. Assuming that an action a_t is taken and the agent moves from belief state $\boldsymbol{s}_t$ to a new state $\boldsymbol{s}_{t+1}$ while receiving a reward r_t, a new action a_{t+1} is chosen according to the current policy:

$$\pi(\boldsymbol{s}_{t+1}) = \arg\max_a Q_t^\pi(\boldsymbol{s}_{t+1}, a). \tag{6.3.1}$$

In that way, the estimated Q value of the new state-action pair is used to update the action-value of the previous state-action pair:

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha\delta_t, \tag{6.3.2}$$

where

$$\delta_t = r_t + \gamma Q_t^{\pi}(s_{t+1}, a_{t+1}) - Q_t^{\pi}(s_t, a_t) \tag{6.3.3}$$

is known as the one step temporal-difference (TD) error. The term α is the learning rate which set to some small value (e.g. $\alpha = 0.01$) and can be occasionally decreased during the learning process.

An additional mechanism that has been employed is that of *eligibility traces*. This allows rewards to back-propagated to recently visited states, allocating them some proportion of the current reward. Every state-action pair in the Q table is given its own eligibility value (e) and every time when the agent visits a particular state-action pair, its eligibility value set equal to 1 (*replacing traces*, [114]). On each step, the eligibility traces for all state-action pairs are decayed by a factor of $\gamma\lambda$, where $\lambda \in [0, 1]$ is the trace decay parameter. The TD error forward proportional in all recently visited state-action pairs as signalised by their non-zero traces according to the following update rules:

$$Q_{t+1}^{\pi}(s, a) \leftarrow Q_t^{\pi}(s, a) + \alpha\delta_t e_t(s, a) \qquad \forall s, \forall a, \tag{6.3.4}$$

where

$$e_t(s, a) = \begin{cases} 1 & \text{if } \; s = s_t \text{ and } a = a_t \\ 0 & \text{if } \; s = s_t \text{ and } a \neq a_t \\ \gamma\lambda e_{t-1}(s, a) & \text{otherwise} \end{cases} \qquad \forall s, \forall a. \tag{6.3.5}$$

The purpose of eligibility traces is to propagate TD-error to the action-values faster so as to accelerate the exploration of the optimal strategy. The specific version, known as SARSA(λ) [118], has been adopted for the learning of the Ms. Pac-Man agent. An overview of the learning scheme of RL-PacMan agent is given in Algorithm 6.1.

6.3.3 Extension: a Monte Carlo Tree Search based Strategy

This section considers the following open issues, met in our proposed agent. The first one is the selection of the fifth feature, s_5, of our state representation, which is of outmost importance. This is because it suggests the most appropriate *target* that should be followed by the agent. Nevertheless, the selection of the best *target* is far from being a trivial task. The second issue is the selection of the most adequate tactic, which determines the behaviour of the RL-PacMan ghost. For example, which is the smallest distance between a ghost and the agent such as the ghost is not considered as threatening.

MCTS [23] is an efficient search method that could be used for discovering the most appropriate *target* at each time step. It is a best-first search method, which is based on

Algorithm 6.1: The SARSA(λ) Algorithm on the RL-PacMan Agent

Initialize: $Q_0(s, a)$ arbitrary and $e(s, a) = 0$ for all s,a.

```
1  begin
2      t = 0;
3      for each episode do
4          Initialize s_t, a_t;
5          repeat for each step
6              Execute action a_t and receive reward r_t ;
7              Observe state s_{t+1} (Sec. 6.3.1);
8              Select action a_{t+1} using policy derived from Q_t (ε-greedy);
9              Calculate temporal difference error, δ_t (Eq. 6.3.3);
10             Update traces (e_{t+1}) and Q-values (Q_{t+1}) ∀s,a (Eqs. 6.3.5, 6.3.4);
11             t = t + 1;
12         until s is terminal;
13     end
14 end
```

a randomized exploration of the search space. A search tree is built iteratively over time until a predefined *computational budget* is reached, i.e., a number of iterations, a time constraint, or a memory limit. Afterwards, the best performed root action is selected, where the root of the tree corresponds to the current agent's position. Each tree node contains two important statistics: the estimated collected return and the number of visits of each node. The MCTS consists of four steps (see, Fig. 6.3), which are executed at each search iteration [29]:

- Selection: Starting from the root node, children are chosen based on a selection policy until a leaf node is reached.
- Expansion: All children are added to the selected leaf node according to the available actions.
- Playout: A simulated playout is executed from the new added node according to a default policy until a terminal state is reached.
- Backpropagation: The result of the simulated playout is backpropagated through the selected nodes up to the root node, updating nodes' statistics.

An ensemble of learning schemes would be considered for the selection of the most appropriate tactic that should be followed by the agent. More specifically, a number of reinforcement learning agents will be trained simultaneously, with each one following its tactic. It is worth noting that a tactic is defined by the proposed representation of the state space. For example, a survival tactic is supposed to be a strategy where ghosts are considered threatening any time. On the other side, in a greedy tactic the agent's

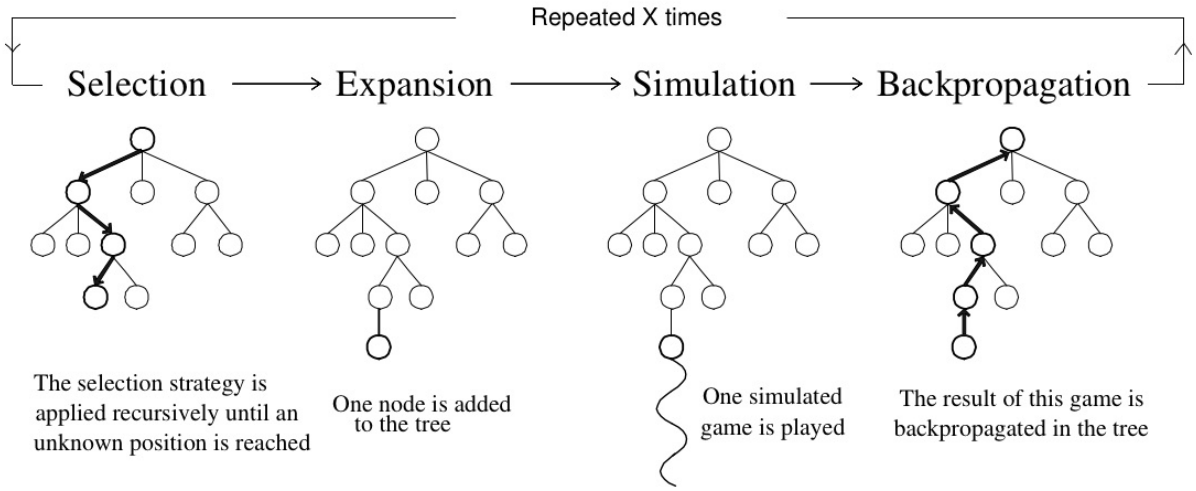


Figure 6.3: Steps of Monte Carlo tree search

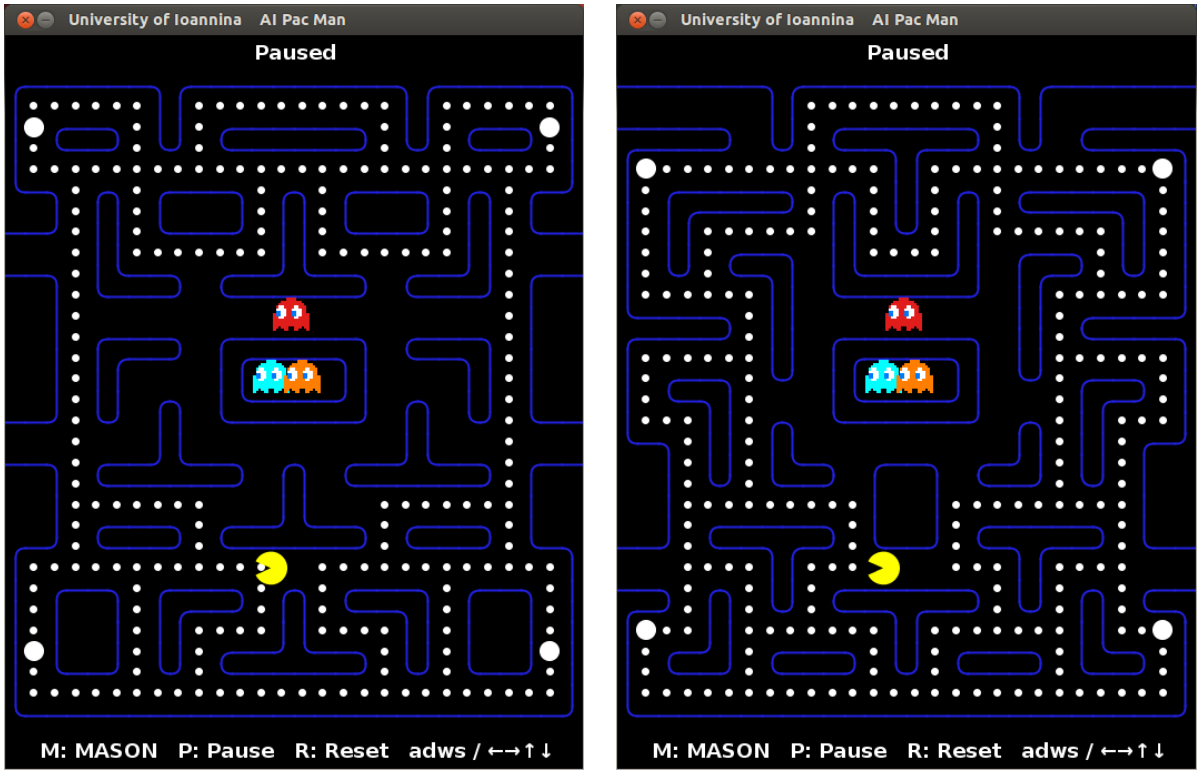
target is to eat as much as possible food even if a ghost is approaching threateningly. In this way, the selection of the most appropriate strategy is translated into a multi-armed bandit problem, where the Upper Confidence Bound (UCB) [7] algorithm can be employed, offering a balance between exploration and exploitation. Thus, the most appropriate strategy is selected and based on this strategy the best move is chosen.

6.4 Empirical Evaluation

A series of experiments has been made for the evaluation of the performance of the proposed RL-PacMan agent at the Ms. Pac-Man domain. All experiments were conducted by using the MASON multi-agent simulation package [79] which provides a faithful version of the original game. Due to the low complexity of the RL-PacMan agent along with its limited requirements on memory and computational resources, the experiments took place on a conventional PC (Intel Core 2 Quad (2.66GHz) CPU with 2GiB RAM).

Three mazes of the original Pac-Man game, illustrated in Figs. 6.1 and 6.4, are used in our experiments. The first one (Fig. 6.1) was used during the learning phase for training the RL agent, while the other two mazes (Fig. 6.4) were applied during the testing process. In all experiments, we have set the discount factor (γ) equal to 0.99 and the learning rate (α) equal to 0.01.

The reward function used in our research is listed in Table.6.4. It must be noted that our method does not show any significant sensitivity to the above reward values. However, a careful selection of the reward values is necessary as the agent's goal is formalized in terms of the reward signal passed from the environment to the agent. Additionally, we consider the Ms. Pac-Man game as an episodic task where an episode is terminated either when all the dots are collected (*win*) or the Ms. Pac-Man is col-



(a) **Light blue maze** (b) **Orange maze**

Figure 6.4: Mazes used for evaluating the proposed RL agent

Table 6.2: The reward function for different game events

Event	***Reward***	***Description***
Step	-0.5	Ms. Pac-Man performed a move in the empty space
Lose	-35	Ms. Pac-Man was eaten by a non-scared ghost
Wall	-100	Ms. Pac-Man hit the wall
Ghost	$+1.2$	Ms. Pac-Man ate a scared ghost
Pill	$+1.2$	Ms. Pac-Man ate a pill

lided with a non-scared ghost. Finally, the performance of the proposed approach was evaluated in terms of four distinct metrics:

- Average percentage of successfully level completion.
- Average number of *wins*.
- Average number of steps per episode.
- Average score attained per episode.

A two-stage strategy has been followed by the RL-PacMan agent during the learning process. At the first phase, the agent is trained without the presence of any ghost in

the maze. In that case, the agent's goal is to eat all the dots as fast as possible. At the second phase, the agent is initialized with the policy discovered at the first phase and the ghosts are inserted into the maze. Likewise, the agent's objective is to clear the maze avoiding the 'non-scared' ghosts and gaining as much points as possible by eating pills and chasing the edible ghosts.

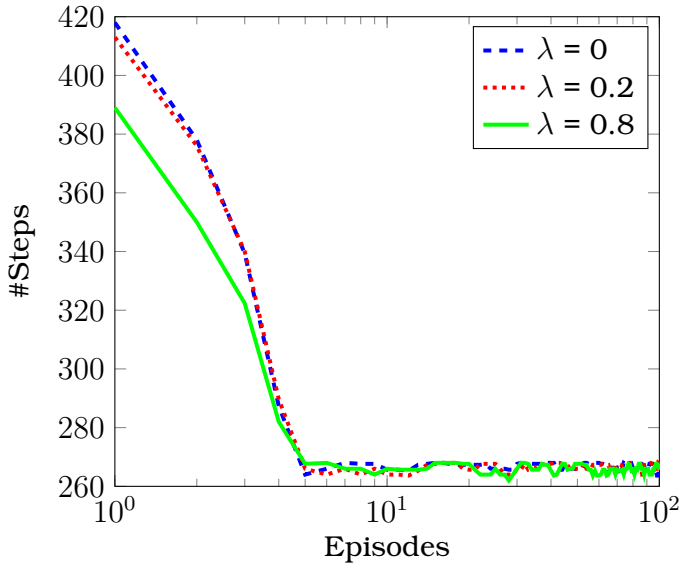


Figure 6.5: Learning progress of the agent at the pink maze without ghosts

Figure 6.5 illustrates the depicted learning curve during the first phase. More specifically, the mean number of steps (after 20 different runs) required by the agent in order to clear all the pills at the first maze (Fig. 6.1) is represented. In order to study the effectiveness of the eligibility trace (Eqs. 6.3.5) to the RL-PacMan agent, three different values $(0, 0.2, 0.8)$ of the trace decay parameter λ are examined. It is worth noting that for each one of these values, the RL-PacMan discovers almost the same policy . Another useful remark is that the learned policies are close enough to the optimal ones as the RL-PacMan agent achieves to eat all of the pills (220) appearing at the maze, by performing only 260 steps (only 15% moves performed in positions with no pills). Nevertheless, it becomes apparent that the value of $\lambda = 0.8$ accelerates the learning process, as our agent achieves to reach an optimal policy quite faster compared to the other two values. For this reason, we set the trace decay parameter, λ, equal to 0.8 for the experiments that follow.

The learning performance of our agent at the second phase is illustrated in Fig. 6.6, in terms of (a) the percentage of level completion and (b) the number of wins (successful level completion) in the last 100 episodes. As it becomes apparent, the RL-PacMan converges quite rapidly at an optimal policy as only 800 episodes are required. At the same, the RL-PacMan agent manages to handle trapped situations with success, completing the level with a high-probability. In our view, the 40% of the level completion suggests a satisfactory playing capability of our agent at the Ms. Pac-Man game.

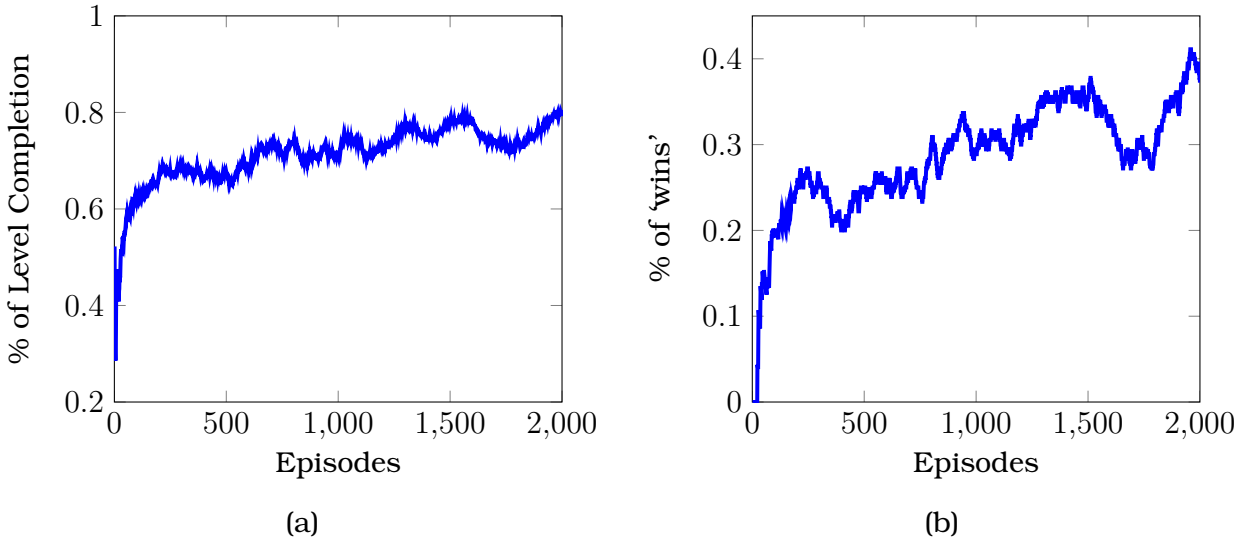


Figure 6.6: Learning progress of the agent at the pink maze with ghosts

For the measurement of the generalization capability of the proposed mechanism, the policy that was discovered at the end of the learning phase tested into two unknown mazes (Fig. 6.4). Table 6.3 depicts the performance of the fixed policy in the three mazes, where the statistics (mean value and standard deviation) of the evaluation metrics were calculated after running 100 episodes. It is interesting to note that the RL-PacMan agent shows a remarkable stability on his behaviour to both unknown mazes providing high generalization abilities.

Table 6.3: Testing performance

Maze	***Level completion***	***Wins***	# ***Steps***	***Score***
Pink (Fig. 6.1)	80% (±24)	40%	348.7 (±153)	2292.3 (±977)
Light blue (Fig. 6.4(a))	70% (±24)	33%	319.4 (±143)	2538.4 (±1045)
Orange (Fig. 6.4(b)	80% (±20)	25%	360.8 (±155)	2515.7 (±1011)

Finally, the obtained policy was tested by playing 50 consecutive games (starting with 3 lives and adding one live at every 10000 points). Table 6.4 summarizes the results where the mean score along with the maximum score found in all three tested mazes are calculated. The particular results verify our previous observations about the generalization ability of the RL-PacMan agent which manages to discover a generic optimal policy allowing the agent to navigate safely at each maze.

Table 6.4: RL-PacMan achieved game score

Mazes	***Average Scores***	***Max Score***
Pink maze (Fig. 6.1)	9665	20860
Light blue maze (Fig. 6.4(a))	12753	38840
Orange maze (Fig. 6.4(b)	11587	27620

6.5 Summary

In this chapter, we have presented the reinforcement learning RL-PacMan agent that learns to play the famous arcade game Ms. Pac-Man. An abstract but informative state space representation has been introduced that allows flexible operation definition possibilities through the reinforcement learning framework. The on-policy, SARSA(λ), reinforcement learning algorithm has been used to discover an optimal policy. The experimental study demonstrates the ability of the RL-PacMan agent to reach optimal solutions in an efficient and rapid way. Finally, it has been proved that providing a proper feature set as input to the learner is of outmost importance as speed up the learning process without the necessity of any function approximation scheme.

Chapter 7

A Bayesian Ensemble Regression Framework on the Angry Birds Game

Physics-based simulation games such as Angry Birds, have received considerable and increasing attention during the last years. They are based on a simulator that has complete knowledge about the physical properties of all objects of the game world. This makes these games quite realistic, as they are able to simulate each move and its consequences to the real world with high precision. Despite the fact that these games are seemingly simple at a first glance, that is far from true. The extremely large or infinite number of the available actions, makes the particular games demanding and simultaneously attractive. The large number of moves stems from the fact that small devations may result in differences in the outcome of the physics simulation. At the same time, the action's outcome is really hard to be predicted in advance without an explicit knowledge of the game's physical properties. In addition, it must be mentioned that the game scene can usually be observed through a vision system taking screenshots which corresponds to how humans are perceiving these games. Consequently, it becomes apparent that a number of issues have arisen which demand new techniques that can investigate the fundamental physical game processes, so as to establish efficient AI agents which will be able to play as good or better than the best human players.

In this chapter, we propose a Bayesian ensemble regression framework for designing an intelligent agent for the Angry Bird domain. The novelty of proposed methodology lies in the construction of an informative encoding scheme of the game scenes, as well as its ability to make accurate predictions and measure the effectiveness of each possible target through a compact ensemble model. These aspects are very important since they manage to build a low complexity agent, being in a form that is suitable for real-time rendering and allowing that to be applicable in a real-time game such as Angry Birds.

Two are the main building blocks of our methodology:

- Firstly, a novel tree structure is proposed for mapping scenes of game levels, where the nodes represent different material of solid objects. More specifically, each tree's node depicts adjacent objects which are constructed by the same material. This scene representation is informative as incorporates all the necessary knowledge about game snapshots, and simultaneously abstract so as to reduce the computational cost accelerating the learning procedure. The specific tree representation allows the construction of an efficient and simultaneously powerful feature space that can be used next during the prediction process.

- Secondly, an ensemble learning approach [86] is designed where every possible pair of 'object material' - 'bird type' has its own Bayesian linear regression model for the estimation of the expected return. In this way, the prediction ability of our scheme becomes much more accurate as it is able to distinguish possible associations between different types of birds and objects' materials. An ensemble integration framework based on the UCB algorithm [7] has employed using the predictions of every regressor, to obtain the final ensemble prediction. After finishing the shot, an online learning procedure is executed in order to adjust the model parameters of each selected regressor.

As experiments indicate, the proposed agent offers both flexibility and robustness and obtains superior modeling solutions. Since there are public available results of all teams participating in the last two years AIBIRDS competitions, we have made comparison with all of them, as well as with the naive agent which is a sufficient baseline for evaluation. In all cases we took excellent results.

The remainder of this chapter is organised as follows. Angry Birds game is presented in Section 7.1. Section 7.2 review a number of related artificial intelligence techniques on the Angry Birds. The general framework of our methodology is described step-by-step in Section 7.3, that consists of the proposed tree structure (Section 7.3.1), together with the Bayesian ensemble mechanism of linear regression models (Section 7.3.4). Furthermore, some issues are discussed about the *feasibility* property of tree nodes (Section 7.3.3), as well as about the *tap timing* decision strategy (Section 7.3.5). In Section 7.4, we assess the performance of the proposed methodology reporting results

obtained by applying our method to the first two levels of the 'Poached Eggs' game set. Finally, Section 7.5 summarizes this chapter.

7.1 The Angry Birds Game

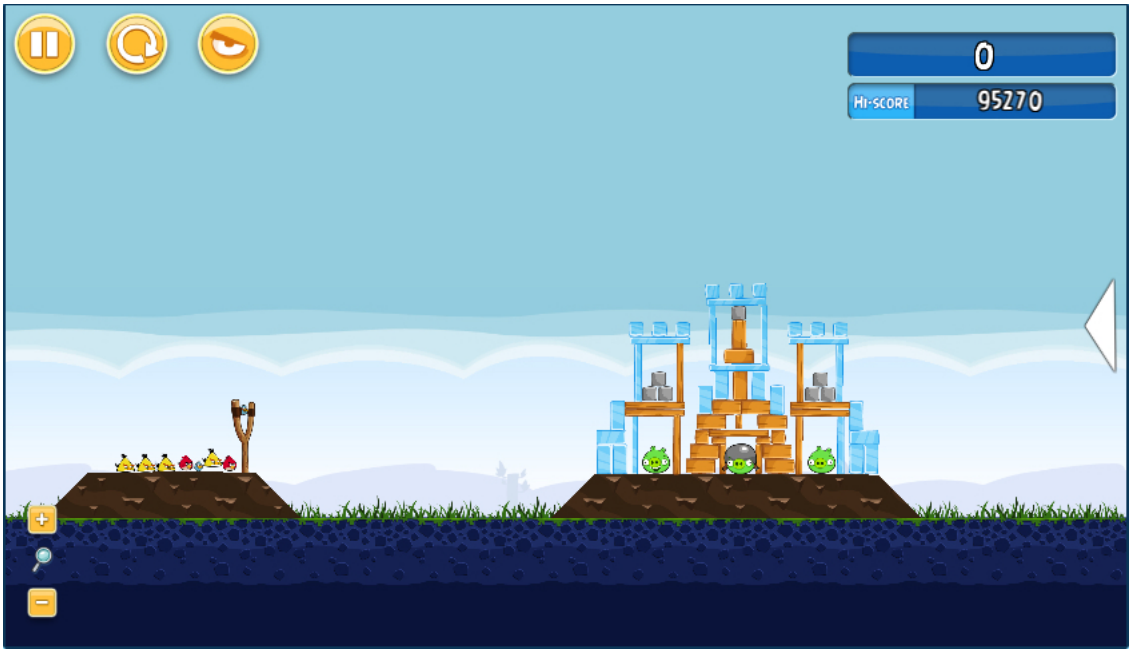


Figure 7.1: A screenshot of the Angry Birds game (21^{st} level at the first series of episodes on the freely available 'Poached Eggs' season)

Angry birds (Fig. 7.1) was first launched in 2008 by Rovio(TM), and since then it has become one of the most popular games nowadays. The objective is to shoot birds using a slingshot in a way that all pigs are killed with the fewest number of used birds. The pigs are usually protected by complicated structures consisting of various types of building materials which must be destroyed for killing pigs. Several types of birds are available, with some of them being more effective against particular materials. At each time step, the point where the bird is released from the slingshot must be selected. In addition, the player have to decide the exact tap time during the flight of the bird where its optional special feature will be activated offering extra power to that. The score (or return) is calculated in terms of the number of pigs killed, the number of the unused birds, as well as the destruction on the structure that achieved after each shot. The fewer birds are used as well as the more damage to the structures achieved, the higher the received score (or return).

Due to its nature (e.g. large state and action spaces, continuous tap timing, various objects' properties, noisy object detection, inaccurate physical models, etc.), Angry Birds constitutes a really challenging task for the development of intelligent agents. At

the same time, the Angry birds competition[1] (AIBIRDS [106]) provides a very attractive venue which offers the opportunity to various AI approaches to compete against each other, as well as to evaluate their performance by playing in unknown game levels. A basic game platform [54] is provided by the organizers, that is based on the Chrome version of the Angry Birds and incorporates a number of components such as, computer vision, trajectory planning and game playing interface which can be freely used. We have also used this platform for developing the proposed agent.

7.2 Artificial Intelligence on Angry Birds

During the last two years, a number of interesting approaches have been proposed which are focused on the development of AI agents with playing capabilities similar to those exhibited by expert human players. These particular works rely on various AI techniques, such as logic programming, qualitative reasoning, advanced simulation, structural analysis, analysis of predicted damage, and machine learning methodologies.

In [157, 147, 51], the qualitative spatial representation and reasoning framework of [30] has been adopted for extracting relationships among scene objects. In [157], an extension of Rectangle Algebra [9] has been proposed for the determination of structure properties, such as its stability or the consequences after some external influences act. On the other hand, in [147] a qualitative physics method has been presented for the examination of the structure properties. In these two works, the action selection was made by measuring all possible shots in terms of a heuristic value function which depends on the shot's influence on the structures. On the other hand, in [51] a decision making under uncertainty scheme was applied for selecting of the most appropriate target according to an utility function.

An alternative work has been presented in [88] that employs the Weight Majority algorithm and the Naive Bayesian Network for selecting the most appropriate shot at each time step. However, a disadvantage on this scheme is that the constructed feature space is extremely large, since it incorporates a large amount of information about the game scene. In addition, it requires a huge amount of training data to be gathered in advance by using a number of different playing agents. Also, an extra effort is needed so as to manually label input data as positive (shots in winning games) and negative (shots in losing games) examples. Another work has been described in [100] based on a model-based methodology for learning the physical model of the world. For this reason, a number of trajectories are evaluated in the approximated model by performing a maximum impact selection mechanism. Last but not least, there are works that combine a limited number of different strategies, some of them really simple, and select the most adequate between them according to the game level, with quite good performance.

[1] https://aibirds.org/

7.3 AngryBER Agent Strategy

The proposed methodology is focused on describing the game scenes with an appropriate and useful tree structure so as to build an efficient state space representation. In addition, a decision making mechanism has been designed using a Bayesian ensemble regression framework that offers robustness and adaptability to dynamically changing situations. Our work is based on the Angry Bird Game Playing software (version 1.32).

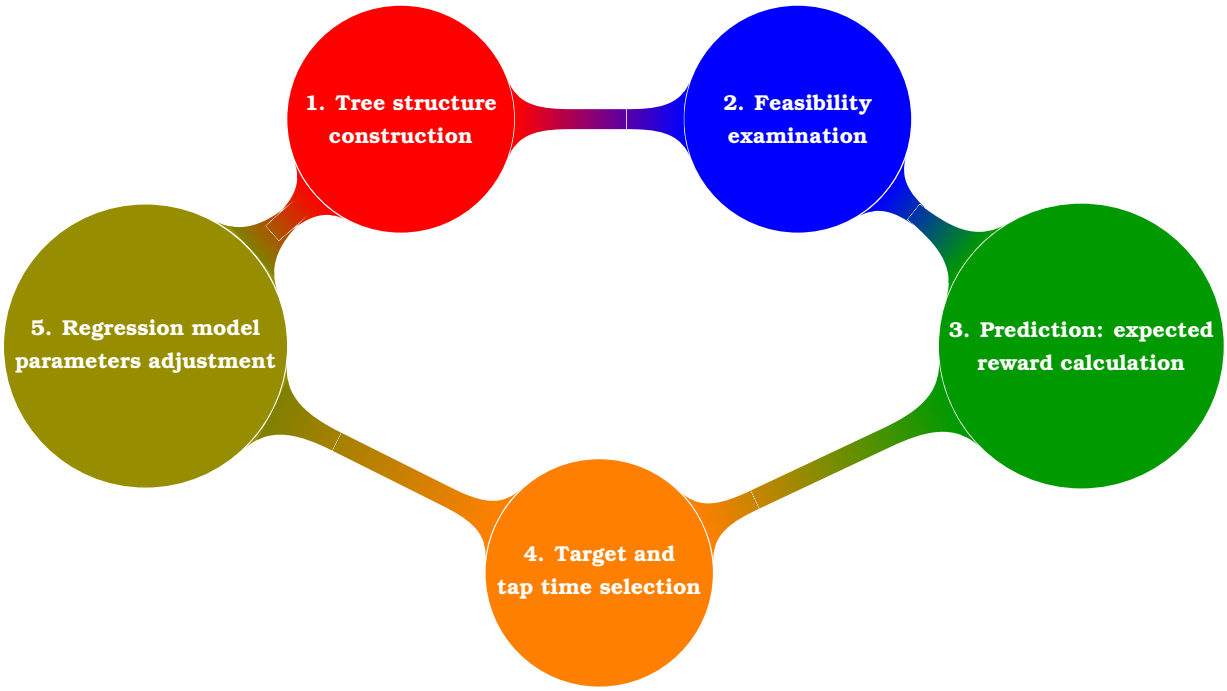


Figure 7.2: Flow diagram of the proposed method

Figure 7.2 illustrates briefly the main building blocks of the proposed approach. A step-by-step description is the following:

1. Construct the **tree structure** of the game scene and establish a feature space.
2. Examine the **nodes feasibility** in terms of their ability to be reached (possible targets).
3. Predict the return of each feasible node (target) according to a **Bayesian Ensemble Regression** scheme, which takes into account the type of the object's material and the available bird. The most appropriate node is then selected as target.
4. **Tap timing** selection and perform shooting.
5. Adjust the model parameters of the selected regressor using an online **learning procedure**.

Next, we give a detailed description of the above parts of our methodology.

7.3.1 A Tree Structure Representation of Game Scenes

The computer vision component of the Angry Birds game is used to analyze the video game scenes. It provides a list of all objects in the scene and identifies information about their type, location, and bounding box. The vision component can recognize the next seven (7) types of objects' materials:

Ice/Glass (**I**)	Wood (**W**)
Stone (**S**)	Rolling Stone (**RS**)
Rolling Wood (**RW**)	Pig (**P**)
TNT (**T**)	

The state space representation of the proposed method is based on the construction of an efficient tree structure in an attempt to arrange and manipulate all the scene objects and their attributes into a compact structure. It consists of a number of nodes that represent different spatial objects of the scene and a number of edges between them that signify their relations.

The proposed tree structure is created through a two-stage process. At first, a complete tree of the game scene is designed by scanning a snapshot in the horizontal direction starting from the ground (level 1). Every time a different object is found (either in material type, or in shape), a new node is added to an appropriate level of the tree. Furthermore, a *virtual* root node is created and connected with all nodes that corresponds to roof objects, i.e. objects that do not have any other object in higher level; see for example nodes s_{11}, s_{15} and s_{91} at Fig. 7.3. The tree structure provides a convenient and attractive layout of the objects relationships, as well as a natural way for handling complex objects. An example is shown in Fig. 7.3 that illustrates the tree of the first game level.

After building the initial tree structure, a tree reduction procedure is performed. During this phase, we traverse the tree and merge nodes of either adjacent, or the same level, in a recursive manner. Merging is done between nodes that have the same material type, are (approximately) adjacent and whose their (vertical or horizontal) sides are equal. Obviously, the merging procedure is not allowed for the object's type of pigs and TNTs. As it is expected, the merging procedure is capable of significantly reducing the size of the tree and therefore the computational cost of the decision making process that follows, since it finally can produce less possible targets for shooting. An example of this phase is shown in Fig. 7.5 for the game scene of Fig. 7.4, where the number of nodes is reduced from 30 (initial phase) to 18. In this example the complete tree nodes $s_{71}, s_{72}, s_{81}, s_{82}$ that belong to the same or adjacent levels of the complete (left) tree, are merged into a single node (s_{51}) in the reduced (right) tree. Similarly, nodes s_{11}, s_{12}, s_{21} are merged together.

It must be noted that in the recursive procedure the direction (vertical or horizontal) for merging nodes does not play any significant role in most of cases, since it leads to the same final solution. However, during the experiments priority is given to the

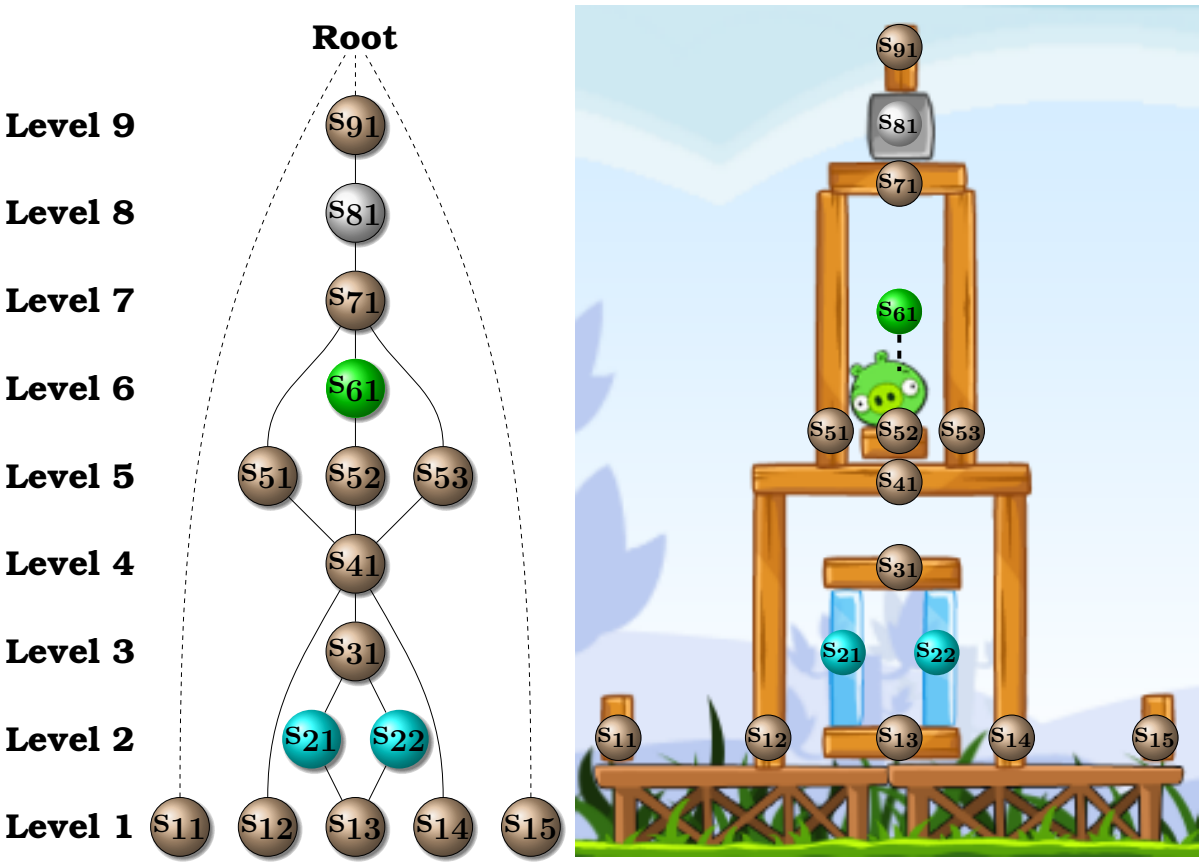


Figure 7.3: The proposed tree structure consisting of 16 nodes at the first game level.

vertical direction. Thus, nodes of adjacent levels (vertical direction) are merged first and in the subsequent step we merge nodes of the same level (horizontal direction). The whole procedure is repeated until no merging can be performed among nodes, see Fig. 7.4. Algorithm 7.1 sketches the main steps for the construction of the proposed tree structure representation.

7.3.2 Feature Extraction

The tree structure framework allows us to extract quantitative features at each node s of the tree. We have selected the following features:

- $x_1(s)$: **Individual weight** calculated as the product of the object's area $Area(s)$ with coefficient c_s whose value depends on the material of the object, i.e. $x_1(s) = Area(s) \times c_s$. All types of objects have the same value for this coefficient, $c_s = 1$, except for Pig (P) and TNT (T) which have a much larger value ($c_s = 10$).

- $x_2(s)$: **Distance (in pixels) to the nearest pig**, normalized to $[0, 1]$ dividing the original distance by a threshold value for the maximum distance (100 in our case).

- $x_3(s)$: **Cumulative weight** calculated as the sum of individual weights of all ancestors $\mathcal{P}(s)$ of the node s in the tree, i.e. $x_3(s) = \sum_{s' \in \mathcal{P}(s)} x_1(s')$.

Algorithm 7.1: The Tree Structure Construction Algorithm

Input : A set of the stucture objects, $\mathcal{O}$
Output : The tree structure, $\mathcal{T}$
Initialize: $k = 1$; $\mathcal{T} = \emptyset$

begin
  Complete tree structure construction phase.;
  while $\mathcal{O} \neq \emptyset$ **do**
    Discover the object with the lowest center to the ground, $o \in \mathcal{O}$;
    Draw a straight horizontal line which passes through its center;
    Find the separated objects intersected by the line, $\mathcal{O}_k$;
    Sort $\mathcal{O}_k$ according to their positions at x-axis;
    Insert $\mathcal{O}_k$ at level $\mathcal{T}_k$, where each object $o \in \mathcal{O}_k$ is added as a separated node;
    $\mathcal{O} = \mathcal{O} \setminus \mathcal{O}_k$;
    $k++$;
  end
  Add *virtual* root node at the highest level k;
  Reduction phase: Merge nodes of the same or adjacent levels according to type and shape properties;
  Extract features from all tree nodes;
  return $\mathcal{T}$;
end

- $x_4(s)$: **Distance from the farthest ancestor**, normalized to $[0, 1]$ by dividing with a threshold value for the maximum height (e.g. 200).

The above feature extraction strategy constructs an abstract but powerful feature space for all possible targets of a game scene. The proposed features are mostly spatial and concern information about geometrical, directional and topological properties of all tree nodes. An example can be seen at Table 7.1 that represents the features values of all 16 nodes of the tree illustrated in Fig. 7.3.

7.3.3 Feasibility Examination

The next step to our approach is to examine each node of the reduced tree structure in terms of its possibility to be reached (possible target). Reachability depends only on the location of objects.

Infeasible situations could be happened in cases where there is a sheltering structure around an object making that not directly reachable to the bird, see for example Fig. 7.6(b). Moreover, it is possible some stable obstacles in the path such as hills, to

(a) (b)

(c) (d)

Figure 7.4: A step-by-step representation of the tree *reduction* procedure at the 16^{th} level of the 'Poached Eggs' season. Each bounding box illustrates an individual object or a group of adjacent objects of the same material, and corresponds to a tree node. (a) Tree nodes that corresponds to the initial tree structure. (b) At the second step, the tree nodes of adjacent levels (vertical direction) are merged. (c) At the third step, the tree nodes of the same level (horizontal direction) are merged. (d) Finally, the tree nodes of adjacent levels are concatenated if it is possible. The tree reduction phase is completed since no nodes are available for concatenation.

block a target (see for example the direct shot at Fig. 7.6(a)). Therefore, an examination step is initially required at each node of the tree so as to ensure that they are reachable.

Two different trajectories are considered: a) a direct shot (angle $<= 45°$) and b) a high arching shot (angle $> 45°$). Both of them are examined to any node of the reduced tree structure in order to estimate their feasibility property. In case where a node can be reached directly from at least one shot it is labeled as *feasible* (Fig. 7.6(a)), otherwise it is considered as *infeasible* and thus cannot be treated as possible target. If both

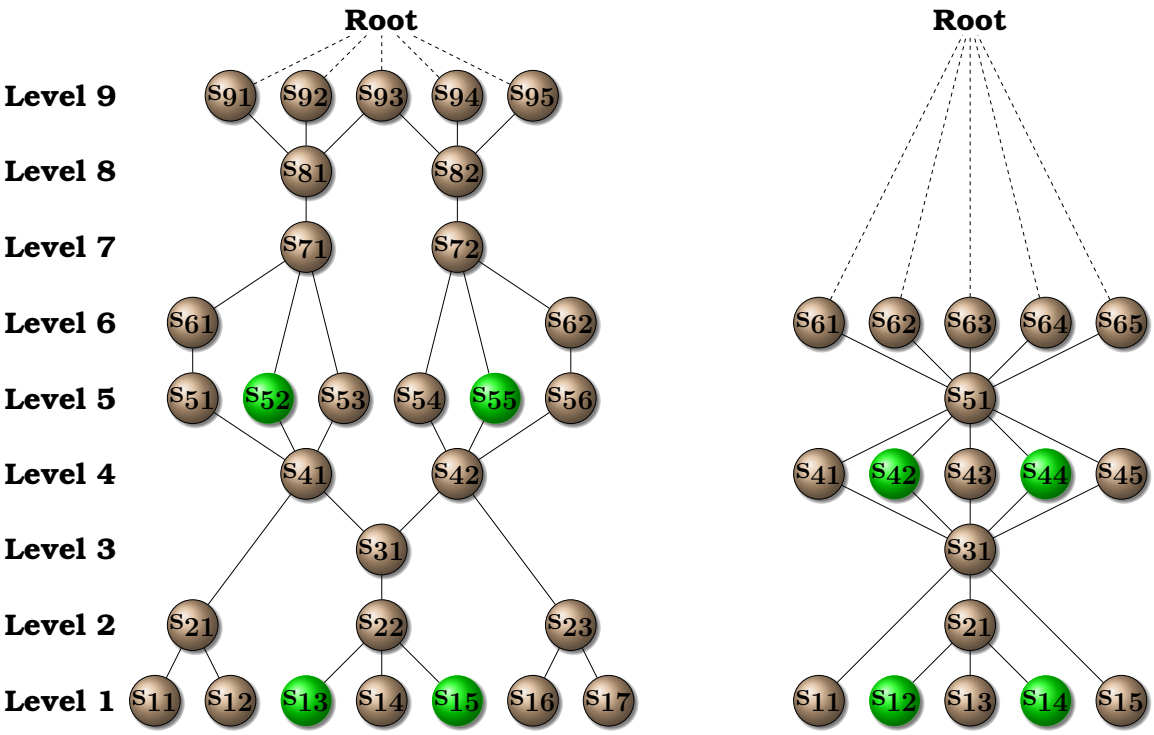
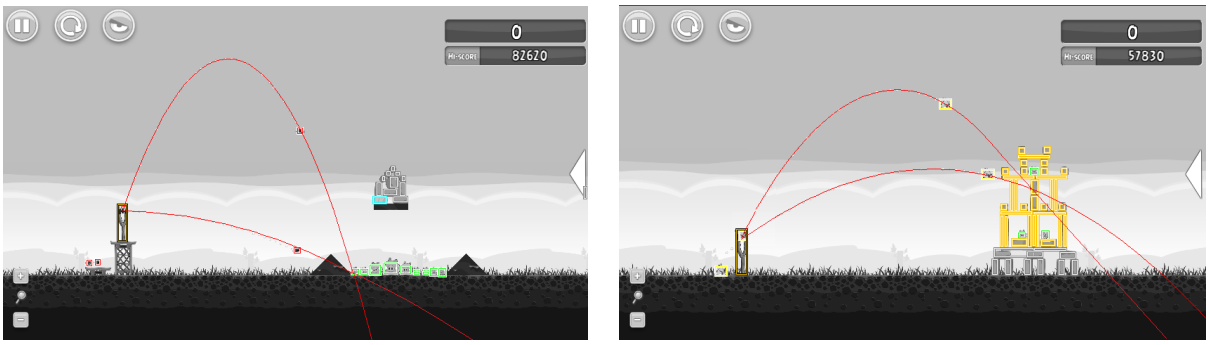


Figure 7.5: Tree structure represention at the 16^{th} level of the 'Poached Eggs' season, before and after tree reduction phase. (a) The initial tree structure (Fig.7.4(a)). (b) The final tree structure (Fig.7.4(d)).



(a) (b)

Figure 7.6: Tree's node feasibility examination. (a) Represents a feasible node (pig) as it is reachable by at least one trajectory. The direct shot is infeasible due to the fact that a hill is interposed between the slingshot and the target. (b) An infeasible node (wood) is represented as it is not directly reachable due to the tree structure.

trajectories are accepted, priority is given on the direct shot due to its effectiveness.

However, there are some special cases which are treated in a different way.

- In the case of the white bird a node is considered as feasible if can be reached by the bird's *egg* (Fig. 7.8), as opposed to the other types of birds.

Table 7.1: The feature vectors along with the feasible and type labels for the 16 tree nodes of Fig. 7.3.

Node	Features					
	Level	Type	$x_1(s)$	$x_2(s)$	$x_3(s)$	$x_4(s)$
			Personal Weight	Pig Distance	Cumulative Ancestors Weight	Farthest Ancestor Distance
s_{11}	1	**W**ood	78	0.818	0	0
s_{12}	1	**W**ood	318	0.501	2467	0.65
s_{13}	1	**W**ood	188	0.660	5532	0.64
s_{14}	1	**W**ood	318	0.501	2467	0.645
s_{15}	1	**W**ood	78	0.818	0	0
s_{21}	2	**I**ce	156	0.504	2623	0.61
s_{22}	2	**I**ce	130	0.504	2623	0.61
s_{31}	3	**W**ood	156	0.341	2467	0.48
s_{41}	4	**W**ood	371	0.151	2096	0.385
s_{51}	5	**W**ood	318	0.164	416	0.36
s_{52}	5	**W**ood	72	0.082	556	0.35
s_{53}	5	**W**ood	318	0.198	416	0.36
s_{61}	6	**P**ig	140	0.170	416	0.32
s_{71}	7	**W**ood	182	0.431	234	0.1
s_{81}	8	**S**tone	156	0.521	78	0.065
s_{91}	9	**W**ood	78	0.651	0	0

- Tree nodes that represent a pig are considered as *feasible* even if the corresponding pig is protected by material objects which are forming its sheltering structure. This is not true only in the case where a steady structure (e.g. hills) protects the pig.

- Finally, the nodes that correspond to objects which are located right from the rightmost pig, rolling stone or rolling wood are characterised as *infeasible* either they are reachable or not.

After the feasibility examination phase, we end up with a set which contains the tree's *feasible* nodes, denoted as $\mathcal{F}$. Only the nodes that belong at $\mathcal{F}$ ($s \in \mathcal{F}$) condidered as possible targets, thereafter. Figure 7.7 illustrates the tree structure at the previously mentioned example (Fig. 7.3) after the feasibility examination ($\mathcal{F} = \{s_{11}, s_{12}, s_{41}, s_{51}, s_{61}, s_{71}, s_{81}, s_{91}\}$), where the *infeasible* nodes are represented by opacity.

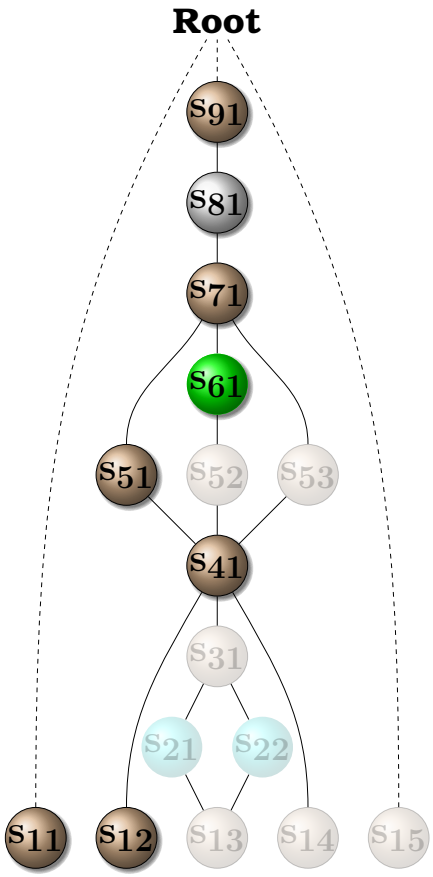


Figure 7.7: The tree structure after the feasibility examination phase at the first game level. The *infeasible* nodes are represented by opacity.

7.3.4 Ensemble of Linear Regression Models

The feasible nodes of the tree structure constitute a set of possible targets of the scene, i.e. points to be hit by a bird. In our approach the task of target selection has been made through an ensemble regression framework. Specifically, as shown previously, each feasible node $s \in \mathcal{F}$ is described with a feature vector $\boldsymbol{x}(s)$. We assume that during the game a sequence of game scores t resulting from each shot are observed. This can be seen as the target attribute that can be modeled using a linear regression scheme of the form:

$$t = \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}(s)) + \epsilon = \sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x}(s)) + \epsilon \ . \tag{7.3.1}$$

In the above equation, M is the order of the regression model and $\boldsymbol{w} = (w_1, \ldots, w_M)^\top$ is the vector of the M unknown regression coefficients. According to this equation, the score is represented as a linearly weighted sum of M fixed basis functions denoted as $\boldsymbol{\phi}(\boldsymbol{x}(s)) = (\phi_1(\boldsymbol{x}(s)), \phi_2(\boldsymbol{x}(s)), \ldots, \phi_M(\boldsymbol{x})(s))^\top$. The error term, ϵ, is assumed to be zero mean Gaussian with variance $1/\beta$, i.e. $\epsilon \sim \mathcal{N}(0, \beta^{-1})$.

To construct the M basis functions we have considered the following strategy: First a number of samples (feature vectors) $\boldsymbol{x}(s)$ has been randomly collected from various

game scenes. Then, an hierarchical agglomerative clustering approach has been performed to them creating an hierarchy of data clusters. In our scheme the standardized Euclidean distance was used as a criterion for merging pairs of clusters. At the end, a number of M clusters were selected from the agglomerative tree whose statistics were used for building the M basis functions and creating a kernel space. In our approach, we have considered normalized Gaussian kernels of the form:

$$\phi_j(\boldsymbol{x}(s)) = \exp\left(-\sum_{r=1}^{D}\frac{(x_r(s)-m_{jr})^2}{2\sigma_{jr}^2}\right) , \tag{7.3.2}$$

where D represents the size of feature space. Also, $\boldsymbol{m}_j = (m_{j1},\ldots,m_{jD})$ and $\boldsymbol{\sigma_j^2} = (\sigma_{j1}^2,\ldots,\sigma_{jD}^2)$ are the mean and variance of the j^{th} cluster, $\forall j = 1,\ldots,M$. It must be noted that the number of clusters M was not so crucial for the performance of the method. During our experimental study, we have found that a number of $M = 150$ clusters was adequate.

Now consider a sequence of n input-target pairs of observations $\{(\boldsymbol{x}_1(s),t_1),\ldots,(\boldsymbol{x}_n(s),t_n)\}$. Given the set of regression model parameter values $\{\boldsymbol{w},\beta\}$ we can model the conditional probability density of targets $\boldsymbol{t}_n = (t_1,\ldots,t_n)$ with the normal distribution, i.e.

$$p(\boldsymbol{t}_n|\boldsymbol{w},\beta) = \mathcal{N}(\boldsymbol{t}_n|\Phi_n\boldsymbol{w},\beta^{-1}I_n) , \tag{7.3.3}$$

where the matrix $\Phi_n = [\boldsymbol{\phi}(\boldsymbol{x}_1(s)),\boldsymbol{\phi}(\boldsymbol{x}_2(s)),\ldots,\boldsymbol{\phi}(\boldsymbol{x}_n(s))]^\top$ of size $n\times M$ is called design matrix and I_n is the identity matrix of order n.

An important issue when using a regression model is how to define its order M (number of basis functions). Models of small order may lead to underfitting, while large values of M may lead to overfitting. One approach to tackle this problem is through the Bayesian regularization framework [129, 17]. According to this scheme, a zero-mean (spherical) Gaussian prior distribution over weights w is considered:

$$p(\boldsymbol{w}|\alpha) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{0},a^{-1}I_M), \tag{7.3.4}$$

where the hyperparameter α is the inverse variance that controls the strength of the prior and I_M is the M-order identity matrix. Thus, the posterior distribution of the weights $\boldsymbol{w}$ is also Gaussian and can be obtained as:

$$p(\boldsymbol{w}|\boldsymbol{t}_n,\alpha,\beta) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_n,\Sigma_n) , \tag{7.3.5}$$

where

$$\boldsymbol{\mu}_n = \beta\Sigma_n\Phi_n^\top\boldsymbol{t}_n \quad\text{and}\quad \Sigma_n = (\beta\Phi_n^\top\Phi_n + aI_M)^{-1}, \tag{7.3.6}$$

are its mean value and the covariance matrix, respectively.

As mentioned previously, we are interested in making predictions for the score value. Suppose we have observed a sequence of n score values $\boldsymbol{t}_n = (t_1,t_2,\ldots,t_n)$. According to the regression model, when examining a feasible node $q\in\mathcal{F}$ of the tree

(possible target) that has a feature vector $\boldsymbol{x}(q)$, we can obtain the posterior predictive distribution of its score t^q which is also Gaussian:

$$p(t^q|\boldsymbol{t}_n, \alpha, \beta) = \mathcal{N}(t^q|\boldsymbol{\mu}_n^\top \boldsymbol{\phi}(\boldsymbol{x}(q)), \sigma_{nq}^2) \ , \tag{7.3.7}$$

where

$$\sigma_{nq}^2 = \beta^{-1} + \boldsymbol{\phi}(\boldsymbol{x}(q))^\top \Sigma_n \boldsymbol{\phi}(\boldsymbol{x}(q)). \tag{7.3.8}$$

This prediction can be used to evaluate a possible target of a game scene by calculating the quantity:

$$\hat{t}^q = \boldsymbol{\mu}_n^\top \boldsymbol{\phi}(\boldsymbol{x}(q)) \ . \tag{7.3.9}$$

However the decision about which is the optimum node to be selected as target depends on the material type of objects, as well as the bird that is available to the slingshot. Therefore, it is reasonable a separate regression estimator for every pair of material type - bird to be used, so as to enhance the accuracy of the decision process. In our approach, we have applied an *ensemble* scheme of regressors, where has been considered that every combination of material type and bird type has its own parametric regression model. Therefore, since there are 7 objects $\times$ 5 birds $= 35$ combinations, we have 35 different linear regression models with parameters $\theta_l = \{\boldsymbol{w}_l, \beta_l\}$, $l = 1, \ldots, 35$. Every time only a subset of these regressors become active based on the type of bird that is available and the type of object's material found in the game scene. The feasible nodes, $q \in \mathcal{F}$, of the tree are then evaluated by calculating the predicted reward value, $\hat{t}^q$, according to Eq. 7.3.9. This is made using the candidate regression model $f(q)$ that suits with the type of object's material and bird type.

The final step before generating the shot is to select the target among all feasible nodes q of the constructed tree. In our approach, we have considered the multi-armed bandit model as the selection mechanism. Specifically, we have employed the Upper Confidence Bound (UCB) strategy [7] which offers a balance between exploration and exploitation during learning. According to the UCB framework we maintain the number of times $n_{f(q)}$ that each arm (regressor $f(q)$) has been played. The decision procedure is made by maximizing the following:

$$q^* = \arg\max_{q \in \mathcal{F}} \left\{ \left(\boldsymbol{\mu}_{n_{f(q)}}^{f(q)} \right)^\top \boldsymbol{\phi}(\boldsymbol{x}(q)) + C \sqrt{\frac{2 \ln N}{n_{f(q)}}} \right\} \ , \tag{7.3.10}$$

where N is the total number of plays so far and C is a tuning parameter of the UCB decision making process that is used to trade off exploration and exploitation (during our experiments we have used $C = 3000$). Intuitively, the UCB framework manages to balance between selecting actions with good belief (targets with large reward prediction) and/or actions which have large uncertainty (small $n_{f(q)}$).

7.3.5 Tap Timing

After the selection of best among the tree's feasible nodes ($\mathcal{F}$), the tap timing procedure is executed. Using the trajectory planner component of the game playing framework

the corresponding tap time is calculated in advance and a tapping is performed right before the estimated collision point. In our approach the tap time strategy depends on the type of birds used:

- Red birds (*Red*) is the leader of the flock, but do not have any special feature at their arsenal. Therefore, there is no need for tapping.
- Blue birds (*the Blues*) split into a set of three similar birds when the player taps the screen. The agent performs a tap in an interval between the 65% and 80% of the trajectory from the slingshot to the first collision object.
- Yellow birds (*Chuck*) accelerate upon tapping which performed between 90% and 95% of the trajectory in the case of high-arching shots (angle $> 45°$). In the case of direct shots (angle $<= 45°$), tap time has been selected randomly between 85% and 90% of the trajectory.
- White birds (*Matilda*) drop *eggs* in the target below them. In this case, tapping is executed when the bird lies above the target (see, Fig. 7.8). As experiments have shown, this strategy is very efficient for handling the specific type of birds.
- Black birds (*Bombs*) are the most powerful member among the birds. No tapping is performed by the agent during the bird flight. The bird blow up himself in a short time period after his impingement with a scene object.

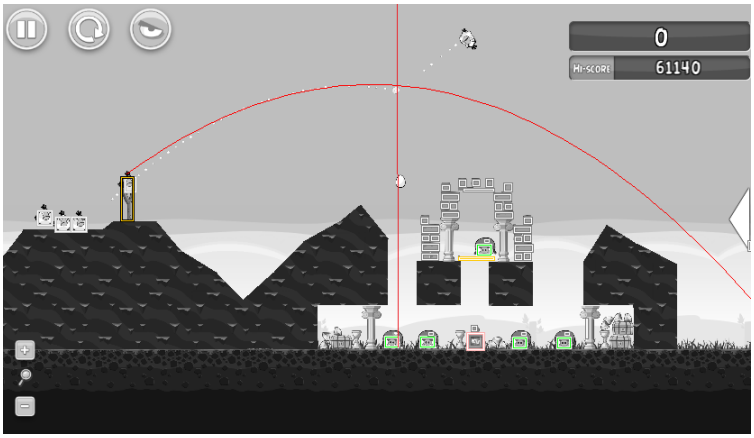


Figure 7.8: Tap timing procedure for the white bird. Tapping is performed only when the bird lies above the target (pig).

7.3.6 Online Learning of Model Parameters

The final step of the proposed scheme is the learning procedure. Due to the sequential nature of data, a recursive estimation framework has been followed for updating the regression model parameters [17]. This can be considered as an online learning solution

to the Bayesian learning problem, where the information on the parameters is updated in an online manner using new pieces of information (rewards) as they arrive. The underlying idea is that at each measurement we treat the posterior distribution of previous time step as the prior for the current time step.

Suppose the tree node q^* has been selected based on the Bayesian ensemble mechanism (Eq. 7.3.10) that corresponds to the regression model $k \triangleq f(q^*)$. Then, the selection frequency, n_k, of this regressor is increased by one, we shoot the target and we receive a score, $t^k_{n_k+1}$. The last constitutes a new observation for the k^{th} regression model, i.e. $\boldsymbol{t}^k_{n_k+1} = (\boldsymbol{t}^k_{n_k}, t^k_{n_k+1})$ which is normally distributed:

$$p(t^k_{n_k+1}|\boldsymbol{w}_k) = \mathcal{N}(t^k_{n_k+1}|\boldsymbol{w}_k^\top \boldsymbol{\phi}(\boldsymbol{x}(q^*)), \beta_k) \ , \tag{7.3.11}$$

where $\boldsymbol{x}_{n_k+1}(q^*)$ is the feature vector of the selected tree node, q^*.

We can now obtain the posterior distribution of weights $\boldsymbol{w}_k$, as:

$$\begin{aligned} p(\boldsymbol{w}^k|\boldsymbol{t}^k_{n_k+1}) &\propto p(t^k_{n_k+1}|\boldsymbol{w}^k)p(\boldsymbol{w}^k|\boldsymbol{t}^k_{n_k}) & (7.3.12)\\ &= \mathcal{N}(\boldsymbol{w}_k|\boldsymbol{\mu}^k_{n_k+1}, \Sigma^k_{n_k+1}) \ , & (7.3.13) \end{aligned}$$

where we can obtain the following recursive forms:

$$\Sigma^k_{n_k+1} = \left[(\Sigma^k_{n_k})^{-1} + \beta_k \boldsymbol{\phi}(\boldsymbol{x}(q^*))\boldsymbol{\phi}(\boldsymbol{x}(q^*))^\top\right]^{-1} \ , \tag{7.3.14}$$

$$\boldsymbol{\mu}^k_{n_k+1} = \Sigma^k_{n_k+1}\left[\beta_k \boldsymbol{\phi}(\boldsymbol{x}(q^*))\boldsymbol{t}^k_{n_k+1} + (\Sigma^k_{n_k})^{-1}\boldsymbol{\mu}^k_{n_k}\right] \ . \tag{7.3.15}$$

The above equations constitute an efficient recursive procedure for adjusting the model parameters of the winner regressor k, after shooting. That provides also the opportunity to monitor learning process. In the beginning of the game, (i.e. step 0) all the information we have about the parameters of all regression models, is the prior distribution $p(\boldsymbol{w}_k)$ which is assumed to be zero mean Gaussian ($\boldsymbol{\mu}^k_0 = \mathbf{0}$) with spherical covariance matrix ($\Sigma^k_0 = a^{-1}I_M$).

Algorithm 7.2 summarizes the basic steps of the proposed method for playing the Angry Bird game.

7.4 Empirical Evaluation

A series of experiments has been conducted in an effort to analyse the performance of the proposed agent (**AngryBER**) in the Angry birds domain. Due to the low complexity of the general framework where our agent is built up, the experiments have taken place in a conventional PC[2]. The source code of the agent can be found in [140]. Our analysis has concentrated mainly on the first 2 episodes from the freely available 'Poached Eggs' season of the Angry Birds game. Each one of the episodes consists of 21 levels, which have to be passed, in order to assume that the episode is successfully completed.

[2]Intel Core 2 Quad (2.66GHz) CPU with 4GiB RAM

Algorithm 7.2: AngryBER Learning Algorithm

while *available bird on the slingshot* **do**
- Game scene's objects detection;
- Tree structure construction of the scene and feature extraction (Alg. 7.1);
- Tree nodes feasibility examination, $\mathcal{F}$ (Sec. 7.3.3);
- Select the target node with the maximum expected return among the feasible nodes, $q^* \in \mathcal{F}$ (Eq. 7.3.10). This corresponds to the regressor $k \triangleq f(q^*)$.
- Compute the most effective timing for the tapping execution (Sec. 7.3.5);
- Shot the target and receive reward, t_{n_k+1};
- Adjust the model parameters (Eqs. 7.3.14, 7.3.15) of the selected regressor, k;

end

The following procedure was used for training the AngryBER agent. Ten (10) complete passes of the previously mentioned episodes have been sequentially executed. The agent remains at the same level if he fails to destroy all pigs found in the game scene. On the other hand, for evaluating our agent we have tried to comply with the AIBIRDS competition rules[3]. Therefore, the agent has at his disposal at least 3 minutes on average in order to complete a game level, corresponding to a total time of 63 minutes for each episode. It must be noted that the results have shown that our agent needs only a part of the available time for a successful episode completion.

In our experiments, we examine two variations of the AngryBER agent, named '**AngryBER$_1$**' and '**AngryBER$_2$**', respectively. The only difference between the two versions lies in the consideration of a diverse set of features (Sec. 7.3.2) that are used to represent the feature space of each tree node. More specifically, both of them use the first two distinctive features $\{x_1, x_2\}$ that are referred to node's personal weight and its distance from the nearest pig, respectively. However, they differ in the way they examine the relation of nodes with their ancestors. Roughly speaking, the first variation '**AngryBER$_1$**' takes into account the density of the structure lies above each node (x_3), while the second '**AngryBER$_2$**' considers the height of the structure located above object materials.

In order to compare the AngryBER agent, we have used the *naive* agent provided by [54], which uses an unsophisticated strategy. More particularly, the *naive* agent shoots the birds directly to the pigs without any further reasoning. It is shown that naive agent provides a sufficient baseline for agent's evaluation. Moreover, we have compared our agent with the agents proposed by all teams participated in 2013 and 2014 AIBIRDS competitions[4] (30 teams in total). It is worth mentioning that for the $2nd$

[3]AIBIRDS 2014 Competition Rules, `https://aibirds.org/angry-birds-ai-competition/competition-rules.html`

[4]AIBIRDS 2014 Benchmarks, `https://aibirds.org/benchmarks.html`

episode only results of the last year teams (2014) are provided (10 teams in total), since an updated version of the vision system was released only last year. This vision system is able to detect the real shape of objects, ground and hills.

We have run 30 independent experiments for each variation of our agent. The results about the first two episodes of 'Poached Eggs' season are presented in Tables 7.2 and 7.3, respectively. In these tables various measures of descriptive statistics about the score reached per level are provided, in an effort to obtain a more comprehensive comparison study. These are: the mean, median, minimum and maximum score found (measures of central tendency), as well as the standard deviation and the interquartile range $IQR = Q3 - Q1$, (measures of variability) of scores.

A number of interesting remarks stems from our empirical evaluation:

- The first one and most impressive observation is that both variants of our Angry-BER agent achieve to pass every level with success. While it may seem easy at a first glance, it is far from true as lot of agents fail to most levels, since the degree of difficulty increases continuously with every successfully completed level.

- AngryBER obtains satisfactory scores in the majority of levels. According to the results, the proposed agent manages to reach (26) high scores at the levels of the two episodes: seven (7) and 19 high scores obtained at the levels of the first and second episode, respectively.

- Additionally, our agents achieve to gain '3-stars' in a considerably high percentage of visited levels. '3-stars' provides a baseline that indicates superior performance. Gaining '3-stars' could be considered as a measure of the agent's ability to destroy all pigs by using the least possible number of birds.

- Another interesting remark is that the mean scores of our agents are always better than those of the benchmarks, with a single exception for the first level of the first episode. Furthermore, considering the median statistic it is easily apparent that our agent performs significantly better than half of the agents provided by the benchmarks.

- Finally, robustness is a key feature of our method as indicated from the small values on both variability measurements (standard deviation and interquartile range) in most levels.

Obviously, the performance of both variations of the proposed approach is superior to the majority of existed methods. Nevertheless, the second variant, **AngryBER$_2$**, performs slightly better than the first one, **AngryBER$_1$**. This is more apparent in difficult levels. Moreover, **AngryBER$_2$** has reached 18 out of 26 high scores found by both agents, and has gained '3-stars' at 34 out of 42 levels in total, while **AngryBER$_1$** has gained '3-stars' at 30 out of 42 levels. Thus, we conclude that the consideration of the height of the structure lying above a tree node (feature, x_4) seems to be more effective than the information of the structure's density (feature, x_3).

Another impressive characteristic of the proposed scheme is its ability to speed-up the learning process and discover near optimal policies quickly. This is attributed to the efficient tree structure representation, in combination with the ensemble learning strategy which allows AngryBER agent to be robust and adaptive to the polymorphic properties of various material-bird types.

Last but not least, the '**AngryBER**$_1$' agent has joined to the 2014 AIBIRDS competition managing to win the second (2^{nd}) price. Our participation in the competition, gave us the opportunity to assess the performance and generalization capability of our agent in unknown challenging levels. As it was proved, the AngryBER agent can cope with success in the most of the assigned levels. The competition results can been found in https://aibirds.org/angry-birds-ai-competition/competition-results.html.

7.5 Summary

In this chapter, we have presented an advanced intelligent agent for playing the Angry Birds game, based on an ensemble of regression models. The key aspect of the proposed method lies on an efficient tree-like scene representation. This allows the exploitation of its superior modeling capabilities to establish a rich feature space. An ensemble scheme of Bayesian regression models is then proposed, where different bird-material type of regressors are combined and act in a competitive fashion. The best prediction is then selected for the decision making process. Learning is achieved in terms of an online estimation framework. Experiments on several game levels demonstrated the ability of the proposed methodology to achieve improved performance and robustness compared to other approaches on the Angry Birds domain.

Table 7.2: Performance statistics at the 21 levels of the first 'Poached Eggs' episode

Level	3 stars	Agent	Mean	Std	Median	IQR	Min	Max
1	32000	**AngryBER$_1$**	29068	±116	29030	0	**29030**	29410
		AngryBER$_2$	28468	±73	28430	80	28430	28800
		Benchmarks	**29233**	±2682	**29635**	1360	18420	**32660**
2	60000	**AngryBER$_1$**	**51196**	±4009	52410	230	33930	53850
		AngryBER$_2$	47363	±6600	**52420**	9080	**34160**	52740
		Benchmarks	47932	±8844	52180	9620	26240	**62370**
3	41000	**AngryBER$_1$**	**41820**	±352	**41910**	0	**40260**	41910
		AngryBER$_2$	41804	±405	**41910**	0	**40260**	41910
		Benchmarks	39029	±4780	41280	1730	24070	**42240**
4	28000	**AngryBER$_1$**	21175	±3932	19190	1470	18720	29150
		AngryBER$_2$	**27471**	±3186	**29010**	1240	**18990**	**29010**
		Benchmarks	23458	±6675	21420	8690	10120	36810
5	64000	**AngryBER$_1$**	**63898**	±3028	**64460**	0	47870	64460
		AngryBER$_2$	63482	±422	63520	0	**61440**	64460
		Benchmarks	60508	±9078	64000	9340	35650	**70350**
6	35000	**AngryBER$_1$**	**34580**	±3222	**35640**	0	**24680**	35720
		AngryBER$_2$	33938	±4864	35385	60	15290	36470
		Benchmarks	23687	±10630	25850	17400	0	**36970**
7	45000	**AngryBER$_1$**	**33763**	±5115	**37000**	6600	**21760**	37140
		AngryBER$_2$	32933	±5874	36940	8690	20550	45780
		Benchmarks	28053	±14166	29350	14830	0	**49120**
8	50000	**AngryBER$_1$**	**47439**	±10407	**54730**	18310	26630	57130
		AngryBER$_2$	40835	±6644	38060	1830	**28710**	55630
		Benchmarks	39473	±13377	43260	20330	0	**57780**
9	50000	**AngryBER$_1$**	32227	±5758	31940	2900	23060	48780
		AngryBER$_2$	**43731**	±4083	**45220**	0	**32450**	48780
		Benchmarks	37300	±11914	38790	21710	0	**51480**
10	55000	**AngryBER$_1$**	46848	±7064	47810	9530	34240	59670
		AngryBER$_2$	**52939**	±8488	49720	13990	**34900**	68110
		Benchmarks	43805	±17857	**50910**	20080	0	**68740**
11	54000	**AngryBER$_1$**	44151	±1719	44860	1050	35610	44860
		AngryBER$_2$	**51347**	±2092	**51425**	1520	**43220**	53580
		Benchmarks	45052	±14644	48390	14780	0	**59070**
12	45000	**AngryBER$_1$**	**52352**	±3018	**53690**	1300	**39680**	54980
		AngryBER$_2$	47412	±7790	48985	15970	36990	56910
		Benchmarks	48348	±14306	53230	9540	0	**61070**

13	47000	**AngryBER$_1$**	26739	±5527	26580	8920	19450	39010
		AngryBER$_2$	**37725**	±7541	**37520**	11080	**21760**	49240
		Benchmarks	30125	±14480	31425	15150	0	**50360**
14	70000	**AngryBER$_1$**	60955	±6002	**65640**	10000	45640	**65640**
		AngryBER$_2$	**64177**	±3805	**65640**	0	**49370**	**65640**
		Benchmarks	52097	±18770	57805	12970	0	**65640**
15	41000	**AngryBER$_1$**	40549	±6401	39350	12360	31190	50020
		AngryBER$_2$	**44908**	±4756	**47370**	7050	**32480**	49620
		Benchmarks	36111	±16019	41390	13660	0	**55300**
16	64000	**AngryBER$_1$**	54554	±5612	52610	4390	45540	**70590**
		AngryBER$_2$	**65191**	±6622	**69590**	7570	**47540**	69590
		Benchmarks	47720	±22513	55600	11470	0	66570
17	53000	**AngryBER$_1$**	**47161**	±3391	**46850**	2050	**39090**	**55760**
		AngryBER$_2$	42631	±3360	42935	5510	37400	49900
		Benchmarks	39260	±16278	44970	8570	0	54750
18	48000	**AngryBER$_1$**	42668	±3961	43305	6810	36350	49790
		AngryBER$_2$	**51669**	±5297	**54180**	7380	**38860**	**56440**
		Benchmarks	35704	±18691	43435	10370	0	54500
19	35000	**AngryBER$_1$**	33681	±3763	35530	6750	**27570**	38090
		AngryBER$_2$	**35461**	±2817	**36690**	2910	25420	39220
		Benchmarks	25043	±15020	30425	20120	0	**40100**
20	50000	**AngryBER$_1$**	46094	±6784	45470	14500	36060	55590
		AngryBER$_2$	**50124**	±6517	**54040**	8250	**37170**	**58550**
		Benchmarks	27268	±21750	36995	43730	0	56050
21	75000	**AngryBER$_1$**	63121	±6029	62395	8190	50740	74240
		AngryBER$_2$	**63551**	±6091	**63000**	8110	**52370**	**76140**
		Benchmarks	34232	±32956	53795	65190	0	75870

Table 7.3: Performance statistics at the 21 levels of the second 'Poached Eggs' episode

Level	3 stars	Agent	Mean	Std	Median	IQR	Min	Max
1	60000	**AngryBer$_1$**	51449	±5559	52635	9960	42770	61470
		AngryBer$_2$	**54363**	±5392	**56880**	7490	**43950**	**64820**
		Benchmarks	38922	±27610	48120	59990	0	64050
2	60000	**AngryBer$_1$**	52769	±2865	52795	3030	47140	59880
		AngryBer$_2$	**53203**	±2747	**53205**	3270	**47650**	58640
		Benchmarks	19510	±33934	0	45790	0	**96180**
3	102000	**AngryBer$_1$**	96909	±6007	98165	8070	86510	114160
		AngryBer$_2$	**100960**	±8532	**99240**	9400	**87730**	**116510**
		Benchmarks	69156	±47915	96375	99490	0	108510
4	50000	**AngryBer$_1$**	50784	±7366	**54540**	10420	26490	58910
		AngryBer$_2$	**51685**	±5862	52290	7680	**32230**	**59690**
		Benchmarks	26018	±27511	24290	52620	0	56550
5	80000	**AngryBer$_1$**	**84366**	±5676	**84775**	6410	**69730**	**93910**
		AngryBer$_2$	82075	±6506	83800	7720	66650	91320
		Benchmarks	43862	±37933	67440	74940	0	78810
6	62000	**AngryBer$_1$**	59301	±4998	57905	8230	51270	69500
		AngryBer$_2$	**60641**	±6010	**58765**	7470	**53990**	**72480**
		Benchmarks	34275	±24892	41640	56750	0	58270
7	50000	**AngryBer$_1$**	42047	±5583	43430	9710	**33260**	53740
		AngryBer$_2$	**48265**	±6839	**47980**	8960	31480	**62140**
		Benchmarks	38245	±20456	46855	8580	0	53450
8	53000	**AngryBer$_1$**	**49397**	±5494	47915	9550	**39470**	58380
		AngryBer$_2$	48942	±6170	**48325**	10840	38210	**59260**
		Benchmarks	34706	±24160	47340	50300	0	57300
9	28000	**AngryBer$_1$**	23239	±3288	22635	2370	19150	34430
		AngryBer$_2$	**24416**	±3274	**23340**	3620	**20420**	33310
		Benchmarks	21007	±13517	22650	6380	0	**46240**
10	40000	**AngryBer$_1$**	**39714**	±3674	**40720**	2280	**32940**	**49910**
		AngryBer$_2$	35459	±1517	35610	1170	30550	41260
		Benchmarks	28863	±16553	35535	16920	0	43600
11	69000	**AngryBer$_1$**	86921	±9421	**86490**	12830	71780	**103620**
		AngryBer$_2$	**87175**	±9486	85725	18050	**74270**	101150
		Benchmarks	23159	±38533	0	53470	0	90540
12	60000	**AngryBer$_1$**	46768	±5216	**46455**	3420	**37300**	**62340**
		AngryBer$_2$	**47335**	±5644	45785	5010	36820	61730
		Benchmarks	18796	±20255	15785	35880	0	46720

13	70000	**AngryBer$_1$**	68954	±7582	70825	12270	50440	78450
		AngryBer$_2$	**73665**	±6492	**72945**	9190	**58010**	**85660**
		Benchmarks	49398	±34248	67425	71970	0	77000
14	50000	**AngryBer$_1$**	40646	±4143	41765	6000	33220	47790
		AngryBer$_2$	**46955**	±5308	**46685**	8170	**33580**	**56710**
		Benchmarks	27376	±19612	35275	43670	0	45230
15	50000	**AngryBer$_1$**	**38923**	±6883	**39325**	8880	28270	52590
		AngryBer$_2$	38473	±7922	38390	14040	**28390**	**53780**
		Benchmarks	5935	±12531	0	0	0	31120
16	62000	**AngryBer$_1$**	53034	±3972	53435	4020	44160	60280
		AngryBer$_2$	**54949**	±4099	**55245**	4480	**46450**	**66500**
		Benchmarks	5614	±17753	0	0	0	56140
17	36000	**AngryBer$_1$**	29289	±2837	28535	5120	25350	35080
		AngryBer$_2$	**29773**	±2258	**29205**	2390	**27050**	**35610**
		Benchmarks	2944	±9310	0	0	0	29440
18	60000	**AngryBer$_1$**	49648	±7015	47945	8930	39910	67960
		AngryBer$_2$	**51621**	±6740	**51020**	6810	**40400**	**69100**
		Benchmarks	9734	±20795	0	0	0	55800
19	47000	**AngryBer$_1$**	38727	±5941	39550	6030	26890	54390
		AngryBer$_2$	**42745**	±7840	**40090**	11540	**32190**	**61520**
		Benchmarks	8335	±17668	0	0	0	45580
20	52000	**AngryBer$_1$**	47751	±8239	47130	**16360**	32990	**56630**
		AngryBer$_2$	**52784**	±5267	**55150**	10	**37770**	55870
		Benchmarks	0	±0	0	0	0	0
21	75000	**AngryBer$_1$**	**70323**	±8395	**70500**	9910	**56890**	**90420**
		AngryBer$_2$	68808	±7316	67410	**10300**	53770	87790
		Benchmarks	6849	±21658	0	0	0	68490

Chapter 8

Conclusions and Future Work

The main objective of this dissertation is the development, implementation and evaluation of machine learning techniques for intelligent agents, which are able to act in unknown environments. Specifically, we elaborate on: i) approximate reinforcement learning, ii) Bayesian reinforcement learning, and iii) AI in games. In Section 8.1 we summarize the novel aspects of the current research on all the aforementioned axed and in Section 8.2 we discuss some possible future work directions.

8.1 Concluding Remarks

In the first part of this thesis, we have studied the approximate reinforcement learning problem in continuous environments. In Chapter 2, an online Bayesian kernelized reinforcement learning algorithm, called as RVMTD, has been presented for the policy evaluation problem. The basic idea of this algorithmic scheme is the transformation of the policy evaluation problem into a linear regression problem. A sparse Bayesian methodology has been used for the estimation of the regression model parameters. An online sparsification procedure has been also adopted, which is based on an incremental construction of a dictionary that contains the most representative states, rendering our methodology practical in large scale continuous domains. Moreover, an extension of the proposed algorithm has also been presented for the learning of the action-value function, allowing us to perform model-free policy improvement.

In Chapter 3, we have presented a model-based reinforcement learning technique for the control learning problem, which is based on an online clustering scheme. More specifically, an online extension of the classical EM algorithm has been adopted, which

is able to handle sequentially arrived samples. Thus, a number of clusters is created, where each cluster encloses data samples with similar identities and hence a dictionary of features is dynamically constructed for modeling the value function. The partitioning of the input space into a finite number of clusters allows the proposed scheme to keep statistics about the model of the environment. Therefore, having approximated sufficiently the dynamics of the environment, the action-value function can be approximated by using the least-squares solution.

In the second part of this thesis, our study has been focused on the Bayesian reinforcement learning problem, where two different Bayesian techniques for reinforcement learning was proposed. Specifically, in Chapter 4 we have proposed a simple linear Bayesian approach to tackle reinforcement learning problems. It has been shown that with an appropriate basis, a Bayesian linear Gaussian model is sufficient for accurately estimating the system dynamics, and in particular when correlated noise is allowed. Policies are estimated by first sampling a transition model from the current posterior, and then, performing approximate dynamic programming on the sampled model. This form of approximate Thompson sampling results in good exploration in unknown environments. The specific approach can also be seen as a Bayesian generalisation of least-squares policy iteration, where the empirical transition matrix is replaced with a sample from the posterior.

In Chapter 5, an online tree-based Bayesian approach for reinforcement learning has been proposed. For inference, we employ a generalised context tree model. This defines a distribution on multivariate Gaussian piecewise linear models that can be updated in closed form. The tree structure itself is constructed using the cover tree method, which is proved to be efficient in high dimensional spaces. Our formulation combines the model with Thompson sampling and approximate dynamic programming to obtain effective exploration policies in unknown environments. The flexibility and computational simplicity of the model renders it suitable for many reinforcement learning problems in continuous state spaces. Actually, we have demonstrated this reasoning in an experimental comparison with a Gaussian process model, a linear model and simple least-squares policy iteration algorithm.

In the third part of this thesis, we have focused on the development of efficient machine learning techniques for two famous video games, the Ms. PacMan and the Angry Birds. In Chapter 6, an intelligent agent for the Ms. PacMan game has been presented, which is based on an on-policy reinforcement learning algorithm for online decision making. The reinforcement learning scheme has been adopted as has been seen to be quite sufficient for designing intelligent agents, able to act in real time. Nevertheless, the high dimensionality of the state spaces encountered in most game domains are a significant barrier that makes them impractical. In this direction, we have presented an approach that is able to deal with the large dynamical environment of the Ms. PacMan game. More specifically, an abstract and at the same time informative state space description has been demonstrated, which is of central interest for the

design of an efficient RL agent. This inference constitutes the main contribution of our study as has been shown that a careful design of the state space representation renders the basic reinforcement learning algorithms practical in large scale domains.

Finally, in Chapter 7 we have introduced the architecture of the AngyBER agent on the Angry Birds domain, where a Bayesian ensemble inference mechanism is suggested for decision making. More specifically, the proposed agent is based on an efficient tree structure for encoding and representing game screenshots, by exploiting its enhanced modeling capabilities. This approach has the advantage of establishing an informative feature space. Thus, the task of game playing is translated into a regression analysis problem. A Bayesian ensemble regression framework has also been presented, by considering the fact that each combination of objects material and bird type has its own regression model. Therefore, we have addressed the problem of action selection as a multi-armed bandit problem, where the Upper Confidence Bound (UCB) strategy is used. An efficient online learning procedure has also been developed for training the regression models. The proposed AngryBER agents have been evaluated on several challenging game levels and their performance have been compared with the results of the agents appeared in the 2013 and 2014 Angry Birds AI competitions.

8.2 Directions for Future Work

In the following, we present some interesting directions for future research that elaborate on a number of open issues related to the methodologies presented in this thesis.

In the sparse Bayesian kernelized reinforcement learning algorithm described Chapter 2, the linear model parameters are estimated by using the relevance vector machine sparse Bayesian methodology. So far, numerous regression schemes have been used in order to efficiently solve the policy evaluation reinforcement learning problem [72, 94, 64]. In this context, different sparse regression scheme, such as Lasso, SVM, etc., could be used for the value function approximation of our scheme. Another research extension could be the incorporation of the eligibility trace mechanism in the proposed temporal difference learning scheme, obtaining a general method with a better learning ability. Furthermore, different exploration schemes may be considered. In our scheme the kernel width is set *a priori* and is equal for each sample inserted in the dictionary. Thus, in our future plans, we aim to examine the case where each dictionary sample has its own kernel width. In this way, we expect to increase the generalization capability of the proposed scheme.

In the model-based reinforcement learning scheme presented in Chapter 2, the value function is represented with the functional form of a linear model. This is achieved by using a number of basis function, which are constructed according to the structure of the clusters. Consequently, a regularized least-squares scheme, such as Lasso or even Bayesian sparse methodologies can be employed in order to eliminate the problem of overfitting. In light of this, we intend to improve the generalization capability of

the proposed learning scheme. During our empirical analysis, we have noticed the tendency of the online EM algorithm to create a large number of clusters. Therefore, another interesting direction for future work could be the adoption of a mechanism that is be able to merge clusters with similar intrinsic structures on their. Another interesting direction, could be the clustering based on the model topology as well as the value function, as proposed in [81].

In the context of the linear Bayesian reinforcement learning approach presented in Chapter 4, Thompson sampling could be used with other Bayesian models for continuous state spaces. Thus, we could be to move to a non-parametric model, for example, to replace the multivariate linear model with a multivariate Gaussian process. The major hurdle would be the computational cost. Consequently, as future work, we would like to use a recently proposed methods for efficient Gaussian processes in the multivariate case, such as [4], where convolution processes are used. Other Bayesian schemes for multivariate regression analysis [83] may be applicable as well. Additionally, it would be highly interesting to consider other exploration methods. One example is the Monte-Carlo extension of Thompson sampling used in [40], which can also be used for continuous state spaces. Other approaches, such as the optimistic transition MDP used in [5] may not be so straightforward to adopt to the continuous case. Nevertheless, while these approaches may be costly computationally, we believe that they would be beneficial in terms of performance.

In the cover tree Bayesian reinforcement learning scheme described in Chapter 5, approximate dynamic programming is employed for the selection of a policy. While in practice ADP can be performed in the background while inference is taking place, and although we seed the ADP with the previous solution, a more incremental approach could be used ideally, for this purpose. One interesting idea would be to employ a gradient approach in a similar vein to [34]. An alternative approach would be to employ an online method in order to avoid estimating a policy for the complete space.[1] Such promising approaches include running bandit-based tree search methods, such as UCT [70], on the sampled models. Another interesting direction for future work is to consider more sophisticated exploration policies, particularly for larger problems. Due to the efficiency of the model, it would be possible to compute near-Bayes-optimal policies, by applying the tree search method used by [144]. It would be interesting to examine continuous actions. These actions can be handled efficiently by using both the cover tree and the local linear models, making the next state directly dependent on the action, through an augmented linear model. While optimising over a continuous action space is challenging, recent efficient tree search methods, such as metric bandits [25] may alleviate this problem. An interesting theoretical direction would be to obtain regret bounds for the problem. Perhaps, this could be done by building upon the analysis of [74] for context tree prediction, and that of [91] for continuous MDPs. The statistical efficiency of the method could be improved by considering edge-based (rather

[1] A suggestion made by the anonymous reviewers.

than node-based) distributions on trees, as suggested in [98]. Finally, as the cover tree method only requires specifying an appropriate metric, the method could be applicable to many other problems, including both large discrete problems and partially observable problems. It would be interesting to see if this approach gives good results even in those cases.

An interesting future direction for the reinforcement learning approach at the Ms. PacMan presented in Chapter 6, could be the investigation of different state space representations. More specifically, a number of different features could also be employed for the state space representation, such as the number of remaining pills, the location of power pills, the distance of Ms. PacMan from each ghost, etc. Another promising direction could be the combination of a number of differently oriented policies (e.g., ghost avoidance) that are trained simultaneously. Also, it could be useful to consider more sophisticated reinforcement learning schemes in order to tackle the online decision making problem, by handling continuous state features.

Regarding, the Bayesian ensemble regression framework proposed in the context of the Angry Birds game (Chapter 7), it could be interesting to study its performance in more challenging game levels and test its generalisation capabilities more systematically. Since the tree structure is very informative and general, another future research direction could be the examination of the possibility to enrich the feature space of our model, by incorporating alternative topological features extracted for the proposed lattice structure, as suggested in [157]. A general issue in regression analysis is the way we define the proper number of basis functions. Sparse Bayesian regression offers an elegant solution to the model selection problem, by introducing sparse priors on the model parameters [129], [112], [18]. During the training phase, the coefficients that are not significant are vanished from the model and thus, only a few of them are retained as most significant. This approach constitutes a possible direction for our future study that could improve the proposed methodology. Finally, alternative regression mechanisms could be applied, e.g. Gaussian Processes [104].

BIBLIOGRAPHY

[1] P. Abbeel and A. Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *International Conference on Machine learning (ICML)*, pages 1–8, 2005.

[2] S. Agrawal and N. Goyal. Analysis of Thompson sampling for the multi-armed bandit problem. In *Conference on Learning Theory (COLT)*, pages 39.1–39.26, 2012.

[3] J. S. Albus. *Brains, Behavior and Robotics*. McGraw-Hill, 1981.

[4] M. Alvarez, D. Luengo-Garcia, M. Titsias, and N. Lawrence. Efficient multioutput Gaussian processes through variational inducing kernels. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 25–32, 2011.

[5] M. Araya, V. Thomas, and O. Buffet. Near-optimal BRL using optimistic local transitions. In *International Conference on Machine Learning (ICML)*, pages 97–104, 2012.

[6] J. Asmuth, L. Li, M. L. Littman, A. Nouri, and D. Wingate. A Bayesian sampling approach to exploration in reinforcement learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 19–26, 2009.

[7] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.

[8] P. Auer, T. Jaksch, and R. Ortner. Near-optimal regret bounds for reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 89–96, 2008.

[9] P. Balbiani, J. F. Condotta, and L. F. D. Cerro. A new tractable subclass of the rectangle algebra. In *International Joint Conference on Artifical Intelligence (IJCAI)*, pages 442–447, 1999.

[10] J. Baxter, A. Tridgell, and L. Weaver. Knightcap: A chess programm that learns by combining td(lambda) with game-tree search. In *International Conference on Machine Learning (ICML)*, pages 28–36, 1998.

[11] R. Begleiter, R. El-Yaniv, and G. Yona. On prediction using variable order Markov models. *Journal of Artificial Intelligence Research*, 22:385–421, 2004.

[12] M. Bellemare, J. Veness, and M. Bowling. Bayesian learning of recursively factored environments. In *International Conference on Machine Learning (ICML)*, pages 1211–1219, 2013.

[13] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957.

[14] D. Bertsekas. Dynamic programming and suboptimal control: From ADP to MPC. *European Journal of Control*, 11(4-5):310 – 334, 2005.

[15] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[16] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *International Conference on Machine Learning (ICML)*, pages 97–104, 2006.

[17] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[18] K. Blekas and A. Likas. Sparse regression mixture modeling with the multi-kernel relevance vector machine. *Knowledge and Information Systems (KAIS)*, 39(2):241–264, 2014.

[19] L. Bom, R. Henken, and M. Wiering. Reinforcement learning to train ms. pacman using higher-order action-relative inputs. In *IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning (ADPRL)*, pages 156–163, April 2013.

[20] J. A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2):233–246, 2002.

[21] S. J. Bradtke and A. G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1):33–57, 1996.

[22] R. I. Brafman and M. Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2003.

[23] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012.

[24] E. Brunskill, B. R. Leffler, L. Li, M. L. Littman, and N. Roy. Provably efficient learning with type parametric models. *Journal of Machine Learning Research*, 10:1955–1988, 2009.

[25] S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári. X-armed bandits. *Journal of Machine Learning Research*, 12:1655–1695, 2011.

[26] P. Burrow and S. Lucas. Evolution versus temporal difference learning for learning to play ms. pac-man. In *IEEE Symposium on Computational Intelligence and Games (CIG)*, pages 53–60, Sept 2009.

[27] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška. Online least-squares policy iteration for reinforcement learning control. In *American Control Conference (ACC)*, pages 486–491, 2010.

[28] P. Castro and D. Precup. Smarter sampling in model-based Bayesian reinforcement learning. In *European Conference on Machine Learning (ECML)*, pages 200–214, 2010.

[29] G. M. J.-B. Chaslot, M. H. M. Winands, H. J. V. D. Herik, J. W. H. M. Uiterwijk, and B. Bouzy. Progressive Strategies For Monte-Carlo Tree Search. *New Mathematics and Natural Computation (NMNC)*, 4(03):343–357, 2008.

[30] A. G. Cohn and J. Renz. Qualitative spatial reasoning. In *Handbook of Knowledge Representation*, pages 551–584, 2007.

[31] M. Daswani, P. Sunehag, and M. Hutter. Feature reinforcement learning using looping suffix trees. In *European Workshop on Reinforcement Learning (EWRL)*, pages 11–24, 2012.

[32] R. Dearden, N. Friedman, and S. J. Russell. Bayesian Q-learning. In *AAAI/IAAI*, pages 761–768, 1998.

[33] M. H. DeGroot. *Optimal Statistical Decisions*. John Wiley & Sons, 1970.

[34] M. P. Deisenroth and C. E. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, pages 465–472, 2011.

[35] M. P. Deisenroth, C. E. Rasmussen, and J. Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7-9):1508–1524, 2009.

[36] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39:1–38, 1977.

[37] C. Dimitrakakis. Tree exploration for Bayesian RL exploration. In *International Conference on Computational Intelligence for Modelling, Control and Automation (CIMCA)*, pages 1029–1034, 2008.

[38] C. Dimitrakakis. Bayesian variable order Markov models. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 161–168, 2010.

[39] C. Dimitrakakis. Complexity of stochastic branch and bound methods for belief tree search in Bayesian reinforcement learning. In *International Conference on Agents and Artificial Intelligence (ICAART)*, pages 259–264, 2010.

[40] C. Dimitrakakis. Robust bayesian reinforcement learning through tight lower bounds. In *European Workshop on Reinforcement Learning (EWRL)*, pages 177–188, 2011.

[41] C. Dimitrakakis, N. Tziortziotis, and A. Tossou. Beliefbox: A framework for statistical methods in sequential decision making. `http://code.google.com/p/beliefbox/`, 2007.

[42] F. Doshi-velez. The infinite partially observable markov decision process. In *Advances in Neural Information Processing Systems 22*, pages 477–485, 2009.

[43] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2001.

[44] M. O. Duff. *Optimal Learning Computational Procedures for Bayes-adaptive Markov Decision Processes*. PhD thesis, University of Massachusetts at Amherst, 2002.

[45] Y. Engel, S. Mannor, and R. Meir. Sparse online greedy support vector regression. In *European Conference on Machine Learning (ECML)*, pages 84–96, 2002.

[46] Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with Gaussian process. In *International Conference on Machine Learning (ICML)*, pages 201–208, 2005.

[47] D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

[48] V. F. Farias, C. C. Moallemi, B. V. Roy, and T. Weissman. Universal reinforcement learning. *IEEE Transactions on Information Theory*, 56(5):2441–2454, 2010.

[49] V. F. Farias and B. Van Roy. Tetris: A study of randomized constraint sampling. In *Probabilistic and Randomized Methods for Design under Uncertainty*, pages 189–202. Springer, 2006.

[50] T. S. Ferguson. Prior distributions on spaces of probability measures. *The Annals of Statistics*, 2(4):615–629, 1974.

[51] L. A. Ferreira, G. A. W. Lopes, and P. E. Santos. Combining qualitative spatial reasoning utility function and decision making under uncertainty on the angry birds domain. In *International Joint Conference on Artificial Intelligence*, 2013.

[52] C. Fraley and A. E. Raftery. How many clusters? which clustering method? answers via model-based cluster analysis. *The Computer Journal*, 41:578–588, 1998.

[53] L. Galway, D. Charles, and M. Black. Machine learning in digital games: A survey. *Artificial Intelligence Review*, 29:123–161, 2008.

[54] X. Ge, S. Gould, J. Renz, S. Abeyasinghe, J. Keys, A. Wang, and P. Zhang. Angry birds game playing software, version 1.32, aibirds.org. Technical report, Research School of Computer Science, The Australian National University, 2014.

[55] M. Geist and O. Pietquin. Kalman Temporal Differences. *Journal of Artificial Intelligence Research*, 39:483–532, 2010.

[56] M. Ghavamzadeh and Y. Engel. Bayesian policy gradient algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, pages 457–464, 2006.

[57] A. Girard, C. E. Rasmussen, J. Q. Candela, and R. Murray-Smith. Gaussian process priors with uncertain inputs - application to multiple-step ahead time series forecasting. In *Advances in Neural Information Processing Systems (NIPS)*, pages 529–536, 2002.

[58] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107, 1968.

[59] M. W. Hoffman, A. L. andMohammad Ghavamzadeh, and R. Munos. Regularized least squares temporal difference learning with nested l2 and l1 penalization. In *European Workshop on Reinforcement Learning (EWRL)*, pages 102–114, 2011.

[60] X. Hu and L. Xu. A comparative study of several cluster number selection criteria. In *International Conference on Intelligent Data Engineering and Automated Learning (IDEAL)*, pages 195–202, 2003.

[61] N. Ikehata and T. Ito. Monte-carlo tree search in ms. pac-man. In *IEEE Conference on Computational Intelligence and Games (CIG)*, pages 39–46, Aug 2011.

[62] T. Jacksh, R. Ortner, and P. Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11:1563–1600, 2010.

[63] N. Jong and P. Stone. Kernel-based models for reinforcement learning in continuous state spaces. In *International Conference on Machine Learning (ICML) workshop on Kernel Machines and Reinforcement Learning*, June 2006.

[64] T. Jung and D. Polani. Least squares svm for least squares td learning. In *European Conference on Artificial Intelligence (ECAI)*, pages 499–503, 2006.

[65] T. Jung and P. Stone. Gaussian processes for sample-efficient reinforcement learning with RMAX-like exploration. In *European Conference on Machine Learning (ECML)*, pages 601–616, 2010.

[66] L. Kaelbling, M.L.Littman, and A. Moore. Reinforcement learning: A survey. *Journal of Artificial Inteligence Research*, 4:237–285, 1996.

[67] S. M. Kakade. A natural policy gradient. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS)*, pages 1531–1538. MIT Press, 2002.

[68] E. Kaufmann, N. Korda, and R. Munos. Thompson sampling: An optimal finite time analysis. In *Algorithmic Learning Theory (ALT)*, pages 199–213, 2012.

[69] P. W. Keller, S. Mannor, and D. Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 449–456, 2006.

[70] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning (ECML)*, pages 282–293, 2006.

[71] J. Z. Kolter and A. Y. Ng. Near-Bayesian exploration in polynomial time. In *International Conference on Machine Learning (ICML)*, pages 513–520, 2009.

[72] J. Z. Kolter and A. Y. Ng. Regularization and feature selection in least-squares temporal difference learning. In *International Conference on Machine Learning (ICML)*, pages 521–528, 2009.

[73] G. Konidaris, S. Osentoski, and P. Thomas. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence (AAAI)*, pages 380–385, 2011.

[74] S. S. Kozat, A. C. Singer, and G. C. Zeitler. Universal piecewise linear prediction via context trees. *IEEE Transactions on Signal Processing*, 55(7):3730–3745, 2007.

[75] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[76] M. G. Lagoudakis, R. Parr, and M. L. Littman. Least-squares methods in reinforcement learning for control. In *Methods and Applications of Artificial Intelligence, Second Hellenic Conference on AI, (SETN)*, pages 249–260, 2002.

[77] B. R. Leffler, M. L. Littman, and T. Edmunds. Efficient reinforcement learning with relocatable action models. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI)*, pages 572–577, 2007.

[78] L. Li, M. L. Littman, and C. R. Mansley. Online exploration in least-squares policy iteration. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 733–739, 2009.

[79] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. MASON: A multi-agent simulation environment. *Simulation*, 81(7):517–527, 2005.

[80] S. Mahadevan and M. Maggioni. Proto-value Functions: A Laplacian Framework for Learning Representation and Control in Markov Decision Processes. *Journal of Machine Learning Research*, 8:2169–2231, 2007.

[81] S. Mannor, I. Menache, A. Hoze, and U. Klein. Dynamic abstraction in reinforcement learning via clustering. In *International Conference on Machine Learning (ICML)*, pages 560–567, 2004.

[82] G. J. McLachlan and D. Peel. *Finite mixture models*. Wiley Series in Probability and Statistics, 2000.

[83] G. Mehmet. A bayesian multiple kernel learning framework for single and multiple output regression. In *European Conference on Artificial Intelligence (ECAI)*, pages 354–359, 2012.

[84] M. Meila and M. I. Jordan. Learning with mixtures of trees. *The Journal of Machine Learning Research*, 1:1–48, 2001.

[85] I. Menache, S. Mannor, and N. Shimkin. Basis Function Adaptation in Temporal Difference Reinforcement Learning. *Annals of Operations Research*, 134:215–238, 2005.

[86] J. Mendes-Moreira, C. Soares, A. Jorge, and J. F. de Sousa. Ensemble approaches for regression: A survey. *ACM Computing Surveys*, 45(1):1–10, 2012.

[87] T. P. Minka. Bayesian linear regression. Technical report, Microsoft research, 2001.

[88] A. Narayan-Chen, L. Xu, and J. Shavlik. An empirical evaluation of machine learning approaches for angry birds. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[89] R. Neal and G. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *NATO Advanced Study Institute on Learning in Graphical Models*, pages 355–368, 1998.

[90] D. Ormoneit and S. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49(2-3):161–178, 2002.

[91] R. Ortner and D. Ryabko. Online regret bounds for undiscounted continuous reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1772–1780, 2012.

[92] I. Osband, D. Russo, and B. V. Roy. (More) efficient reinforcement learning via posterior sampling. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3003–3011, 2013.

[93] S. M. Paddock, F. Ruggeri, M. Lavine, and M. West. Randomized Polya tree models for nonparametric Bayesian inference. *Statistica Sinica*, 13(2):443–460, 2003.

[94] C. Painter-wakefield and R. Parr. Greedy algorithms for sparse reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 1391–1398, New York, NY, USA, 2012.

[95] R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 752–759, 2008.

[96] J. Peng and R. J. Williams. Incremental multi-step q-learning. *Machine Learning*, 22(1-3), 1996.

[97] T. Pepels, M. Winands, and M. Lanctot. Real-time monte carlo tree search in ms pac-man. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(3):245–257, Sept 2014.

[98] F. C. Pereira and Y. Singer. An efficient extension to mixture techniques for prediction and decision trees. *Machine Learning*, 36(3):183–199, 1999.

[99] M. Petrik. An analysis of laplacian methods for value function approximation in mdps. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2574–2579, 2007.

[100] M. Polceanu and C. Buche. Towards a theory-of-mind-inspired generic decision-making framework. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[101] P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete Bayesian reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 697–704, 2006.

[102] M. L. Puterman. *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. John Wiley & Sons, New Jersey, US, 2005.

[103] C. E. Rasmussen and M. Kuss. Gaussian processes in reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 751–759, 2004.

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

[105] J. Reisinger, P. Stone, and R. Miikkulainen. Online kernel selection for bayesian reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 816–823, 2008.

[106] J. Renz. Aibirds: The angry birds artificial intelligence competition. In *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence (AAAI)*, 2014.

[107] I. Rexakis and M. G. Lagoudakis. Directed policy search for decision making using relevance vector machines. *International Journal on Artificial Intelligence Tools*, 23(4), 2014.

[108] G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical report, Cambridge University Engineering Department, 1994.

[109] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 3rd edition, 2009.

[110] S. Samothrakis, D. Robles, and S. Lucas. Fast approximate max-n monte-carlo tree search for ms pac-man. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(1):1 –16, June 2011.

[111] B. Scholkopf and A. Smola. *Learning with Kernels*. MIT Press, 2002.

[112] M. Seeger. Bayesian inference and optimal design for the sparse linear model. *Journal of Machine Learning Research*, 9:759–813, 2008.

[113] D. Silver, R. S. Sutton, and M. Müller. Sample-based learning and search with permanent and transient memories. In *International Conference on Machine Learning (ICML)*, pages 968–975, 2008.

[114] S. P. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1-3):123–158, 1996.

[115] W. B. Smith and R. R. Hocking. Wishart variates generator, algorithm AS 53. *Applied Statistics*, 21:341–345, 1972.

[116] A. L. Strehl and M. L. Littman. Online linear regression and its application to model-based reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1417–1424, 2008.

[117] M. Strens. A Bayesian framework for reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 943–950, 2000.

[118] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press Cambridge, USA, 1998.

[119] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[120] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1038–1044. MIT Press, 1996.

[121] C. Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010.

[122] I. Szita. Reinforcement learning in games. In M. Wiering and M. van Otterlo, editors, *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 539–577. Springer Berlin Heidelberg, 2012.

[123] I. Szita and A. Lorincz. Learning to play using low-complexity rule-based policies: Illustrations through ms. pac-man. *Journal of Artificial Intelligence Research*, 30:659–684, 2007.

[124] G. Taylor and R. Parr. Kernelized value function approximation for reinforcement learning. In *International Conference on Machine Learning (ICML)*, pages 1017–1024, 2009.

[125] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3-4):257–277, 1992.

[126] G. Tesauro. Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134(1-2):181 – 199, 2002.

[127] C. Thierry and B. Scherrer. Building controllers for Tetris. *International Computer Games Association Journal*, 32(1):3–11, 2010.

[128] W. Thompson. On the Likelihood that One Unknown Probability Exceeds Another in View of the Evidence of two Samples. *Biometrika*, 25(3-4):285–294, 1933.

[129] M. Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of Machine Learning Research*, 1:211–244, 2001.

[130] M. Tipping and A. Faul. Fast marginal likelihood maximization for sparse bayesian models. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, 2003.

[131] B. Tong and C. W. Sung. A monte-carlo approach for ghost avoidance in the ms. pac-man game. In *International IEEE Consumer Electronics Society's Games Innovations Conference (ICE-GIC)*, pages 1–8, Dec 2010.

[132] B.-B. Tong, C. M. Ma, and C. W. Sung. A monte-carlo approach for the endgame of ms. pac-man. In *IEEE Conference on Computational Intelligence and Games (CIG)*, pages 9–15, Aug 2011.

[133] J. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

[134] N. Tziortziotis and K. Blekas. A bayesian reinforcement learning framework using relevant vector machines. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence (AAAI)*, pages 1820 – 1821, 2011.

[135] N. Tziortziotis and K. Blekas. Value function approximation through sparse bayesian modeling. In *European Workshop on Reinfocement Learning (EWRL)*, pages 128–139, 2011.

[136] N. Tziortziotis and K. Blekas. A model based reinforcement learning approach using on-line clustering. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 1, pages 712–718, 2012.

[137] N. Tziortziotis, C. Dimitrakakis, and K. Blekas. Linear Bayesian reinforcement learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1721–1728, 2013.

[138] N. Tziortziotis, C. Dimitrakakis, and K. Blekas. Cover tree bayesian reinforcement learning. *Journal of Machine Learning Research*, 15:2313–2335, 2014.

[139] N. Tziortziotis, G. Papagiannis, and K. Blekas. A Bayesian Ensemble Regression Framework on the Angry Birds Game. *ArXiv e-prints*, Aug. 2014.

[140] N. Tziortziotis, G. Papagiannis, and K. Blekas. Angryber: An advanced agent for the angry birds game. `https://code.google.com/p/angry-ber/`, 2014.

[141] N. Tziortziotis, K. Tziortziotis, and K. Blekas. Play ms. pac-man using an advanced reinforcement learning agent. In *Methods and Applications - 8th Hellenic Conference on AI (SETN)*, pages 71–83, 2014.

[142] T. van Erven, P. D. Grünwald, and S. de Rooij. Catching up faster by switching sooner: a prequential solution to the AIC-BIC dilemma. *arXiv*, 2008. A preliminary version appeared in NIPS 2007.

[143] J. Veness, K. S. Ng, M. Hutter, and M. Bowling. Context tree switching. In *Data Compression Conference (DCC)*, pages 327–336, 2012.

[144] J. Veness, K. S. Ng, M. Hutter, W. Uther, and D. Silver. A Monte-Carlo AIXI approximation. *Journal of Artificial Intelligence Research*, 40:95–142, 2011.

[145] J. Veness, D. Silver, A. Blair, and W. W. Cohen. Bootstrapping from game tree search. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1937–1945, 2009.

[146] N. Vlassis, M. Ghavamzadeh, S. Mannor, and P. Poupart. Bayesian reinforcement learning. In *Reinforcement Learning*, volume 12, pages 359–386. Springer Berlin Heidelberg, 2012.

[147] P. Walega, T. Lechowski, and M. Zawidzk. Qualitative physics in angry birds: first results. In *Symposium on Artificial Intelligence in Angry Birds*, 2014.

[148] T. Wang, D. Lizotte, M. Bowling, and D. Schuurmans. Bayesian sparse sampling for on-line reward optimization. In *International Conference on Machine Learning (ICML)*, pages 956–963, 2005.

[149] C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[150] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, 1989.

[151] S. Whiteson, M. E. Taylor, and P. Stone. Adaptive tile coding for value function approximation. Technical Report AI-TR-07-339, University of Texas at Austin, 2007.

[152] F. Willems, Y. Shtarkov, and T. Tjalkens. The context tree weighting method: basic properties. *IEEE Transactions on Information Theory*, 41(3):653–664, 1995.

[153] W. H. Wong and L. Ma. Optional Pólya tree and Bayesian inference. *The Annals of Statistics*, 38(3):1433–1459, 2010.

[154] J. Wyatt. *Exploration and inference in learning from reinforcement*. PhD thesis, University of Edinburgh. College of Science and Engineering. School of Informatics, 1998.

[155] X. Xu, D. Hu, and X. Lu. Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4):973–992, 2007.

[156] X. Xu, T. Xie, D. Hu, and X. Lu. Kernel least-squares temporal difference learning. *International Journal of Information Technology*, 11(9):54–63, 2005.

[157] P. Zhang and J. Renz. Qualitative spatial representation and reasoning in angry birds: The extended rectangle algebra. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR*, 2014.

APPENDIX A

Mathematical Background

A.1 Bayes' Rule

Let A and B be random variables, then according to the Bayes' theorem:

$$\mathbb{P}(A|B) = \frac{\mathbb{P}(B|A)\,\mathbb{P}(A)}{\int \mathbb{P}(B|A')d\,\mathbb{P}(A')}.$$

A.2 Matrix Identities

Matrix Inversion Lemma (Sherman-Morrison-Woodbury formula)

The matrix inversion lemma states that,

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1},$$

assuming that the relevant inverses exist. Here, A, B, C and D are respectively $n \times n$, $n \times m$, $m \times m$ and $n \times m$ matrices.

Matrix Blockwise Inversion

Let a matrix U be partitioned into a blockwise form as

$$U = \begin{bmatrix} A & B \\ C & D \end{bmatrix}$$

where A and D must be square matrices which are invertible. Then

$$U^{-1} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} M & -MBD^{-1} \\ -D^{-1}CM & D^{-1} + D^{-1}CMBD^{-1} \end{bmatrix}$$

where we have defined, $M = (A - BD^{-1}C)^{-1}$. The quantity M^{-1} is known as *Schur complement* of D in matrix U.

Author's Publications

Journal Publications

J1. N. Tziortziots, G. Papagiannis and K. Blekas, A Bayesian Ensemble Regression Framework on the Angry Birds Game, IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG), 2015 (submitted).

J2. N. Tziortziotis, C. Dimitrakakis and K. Blekas, Cover Tree Bayesian Reinforcement Learning, Journal of Machine Learning Reaserch (JMLR), (15):2313-2335, 2014.

Conference Publications

C1. K. Pandremmenou, N. Tziortziotis, S. Paluri, W. Zhang, K. Blekas, L. P. Kondi and S. Kumar, "Quality Optimization of H.264/AVC Video Transmission over Noisy Environments Using a Sparse Regression Framework", in Visual Information Processing and Communication VI, Proceedings of SPIE-IS&T Electronic Imaging, San Francisco, CA, February 2015.

C2. N. Tziortziotis, G. Papagiannis and K. Blekas, A Bayesian Ensemble Regression Framework on the Angry Birds Game, in the ECAI Symposium on Artificial Intelligence in Angry Birds, Prague, Czech Republic, August 2014. *Second Place on the the Angry Birds AI Competiton 2014*.

C3. N. Tziortziotis, K. Tziortziotis and K. Blekas, Play Ms. Pac-Man using an Advanced Reinforcement Learning Agent, in the 8th Hellenic Conference on Artificial Intelligence (SETN 2014), Ioannina, Greece, May 2014.

C4. C. Dimitrakakis and N. Tziortziotis, ABC Reinforcement Learning, in the 30th International Conference on Machine Learning Learning (ICML 2013), Atlanta, USA, June 2013, MLR W&CP 28(3):684-692.

C5. N. Tziortziotis, C. Dimitrakakis and K. Blekas, Linear Bayesian Reinforcement Learning, in the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013), Beijing, China, August 2013.

C6. K. Pandremmenou, N. Tziortziotis, L. P. Kondi and K. Blekas, Resource Allocation in Visual Sensor Networks Using a Reinforcement Learning Framework, in the 18th IEEE International Conference on Digital Signal Processing (DSP), Santorini, Greece, July 2013.

C7. N. Tziortziotis and K. Blekas, Model-based Reinforcement learning using online clustering, in the 24th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2012), Pireus, Greece, November 2012.

C8. N. Tziortziotis and K. Blekas, An online kernel-based clustering approach for value function approximation. in the 7th Hellenic Conference on Artificial Intelligence (SETN 2012), Lamia, Greece.

C9. N. Tziortziotis and K. Blekas, A Bayesian Reinforcement Learning framework using Relevant Vector Machines, in the 25th International Conference on Artificial Inteligence (AAAI-2011), San Francinco, USA, August 2011.

C10. N. Tziortziotis and K. Blekas, Value Function Approximation through Sparse Bayesian Modeling, in the 9th European Workshop on Reinforcement Learning (EWRL-9), Athens, Greece, September 2011.

Magazine Publications

M1. C. Dimitrakakis, G. Li and N. Tziortziotis, The Reinforcement Learning Competition, Artificial Intelligence (AI) Magazine, 2014.

Short Vita

Nikolaos Tziortziotis was born in Trikala, Greece, in 1984. He received the B.Sc. and M.Sc in Computer Science in 2007 and the 2010, respectively, from the Department of Computer Science and Engineering, University of Ioannina, Greece. Since 2010, he has been a Ph.D. candidate at the Computer Science and Engineering Department of the University of Ioannina, where he is a member of the Information Processing and Analysis research group (I.P.AN.). In the past, he was an exchange Ph.D. student at the Artificial Intelligence Laboratory at the Swiss Federal Institute of Technology Lausanne (EPFL). Also, he was a co-organizer of the revived reinforcement learning competition held in ICML 2013. His research interests are in the areas of machine learning, reinforcement learning and decision theory.