Linear Reinforcement Learning with Options

Kamil Andrzej Ciosek

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Centre for Computational Statistics and Machine Learning

Department of Computer Science

University College London

Original version: March 25, 2015

This version: August 1, 2015

Abstract

The thesis deals with linear approaches to the Markov Decision Process (MDP). In particular, we describe Policy Evaluation (PE) methods and Value Iteration (VI) methods that work with representations of MDPs that are compressed using a linear operator. We then use these methods in the context of the options framework, which is way of employing temporal abstraction to speed up MDP solving. The main novel contributions are: the analysis of convergence of the linear compression framework, a condition for when a linear compression framework is optimal, an in-depth analysis of the LSTD algorithm, the formulation of value iteration with options in the linear framework and the combination of linear state aggregation and options.

"You must compose your life action by action, and be satisfied if each action achieves its own end as best can be: and no one can prevent you from that achievement. 'But there will be some external obstacle.' No obstacle, though, to justice, self-control, and reason. 'But perhaps some other source of action will be obstructed.' Well, gladly accept the obstruction as it is, make a judicious change to meet the given circumstance, and another action will immediately substitute and fit into the composition of your life as discussed."

Marcus Aurelius, Meditations 8.32, translated by Martin Hammond

Contents

Introduction

In the late 20th century and early 21st century a combination of new technical, economic and social phenomena has emerged. First, there has been an exponential decrease in the cost of computation (usually referred to as Moore's law) due to the technical advances and increasing economies of scale in the semiconductor industry. Second, we have witnessed a world-wide spread of connectivity via the Internet, which is already very cheap in the developed world and which has started to penetrate into the world's poorest countries. This connectivity, coupled with low-cost access devices such as smartphones (as of the time of writing, prices of internet-enabled ones begin at £30 without contract) and tablets have lead to an overwhelming explosion in the amounts of data available for processing. Third, increasing economic globalization, the fall of communism in Eastern Europe and Russia and the rise of China as a first-tier economic power have led to increased competition to optimize business processes and models. Fourth, the emergence of the internet-based social network, both personal (Facebook) and professional (LinkedIn) has led to a profound change in the way members of society interact, in addition to providing even more data. Some authors refer to these phenomena as the Information Revolution. These developments have led to the need to provide technologies necessary for digitally processing the new data (known as Big Data). Approaches to this problem emerged from different fields, under terms such as data science, computational statistics and machine learning. This work focuses on what emerged form computer science and is traditionally referred to as machine learning, but the distinction between the areas outlined above is very fuzzy, and the different communities are increasingly converging to a single tool-set.

All natural quantitative science is in principle about deriving useful structure from existing information, which can then be used in situations which are new, but is some ways similar. Machine Learning, in its most general form, can be thought of as a meta-quantitative science in that it provides a process of doing this automatically by means of a computer algorithm (although it has to be said that the current state of technology is that it only works for certain well-delimited tasks; indeed, it would be very difficult to predict when or if automatic algorithms will be able to automate the work done by humans in, say, physics departments). Traditionally, Machine Learning has been divided into three areas. The first is Supervised Learning, where the data we are given has tagging information attached to it, known as labels and the typical problem is to find a concise mapping from data to labels. Then there is Unsupervised Learning, where we are given just a bunch of data without any labels and the task is to find useful structure in the data. There is also the field of Semi-Supervised Learning, which is hybrid of the two where we are given some, but not all, labels and the tasks are the same as in Supervised Learning. However, all the above approaches are largely based on the assumption that data samples are drawn independently from some probability distribution. This approach has the problem that many phenomena that occur in the real world do not have this property. In particular, whenever we have a setting where a machine controlled by a computer is interacting with the world, it is reasonable to believe that the outcome depends on the sequence of interactions and that in general, interactions may have consequences reaching into the feature.

Therefore there is a need to have algorithms that are designed to facilitate interactions between the machine and its environment. The study of such algorithms in the context of Machine Learning is called Reinforcement Learning (although much of the same problems were addressed by the Control and Operations Research communities much earlier, in the context of much smaller data-streams, less complex systems and limited computational power). Customarily, one divides Reinforcement Learning into three approaches, based on successively more complex mathematical models (we do not claim here that this taxonomy is exhaustive, only that it is sufficient for our purposes). First there are bandit algorithms, where we are given a finite set of arbitrary probability distributions (known as bandits), which are fixed but unknown. In the simplest version of the problem, our task at each time step is to choose from which distribution to sample with the goal that the total sum of outcomes from all samples should be maximal. Of course the optimum thing to do is to sample the distribution with the maximal mean and the whole difficulty is that we do not know which one that is. Very good algorithms exist for useful versions of the Bandit problem (see for example the excellent monographs by Sébastien Bubeck; Nicolò Cesa-Bianchi and Rémi Munos [1, 2]). Bandit algorithms have been extensively used in the internet advertising industry and the technology can be described as being mature. However, the major limitation of bandit algorithms is that the model does not include any notion of memory or state – it does matter, unlike in the techniques described in the previous paragraph, in which order we interact with the bandits because it influences the information that we have, but samples from a single bandit are independent. Therefore we need a more complex mathematical tool to describe the notion that the environment changes its state when we interact with it. The classic tool to model such a system is the discrete Markov Decision Process (MDP). It is defined as a set of actions defined over the same state space, each action consisting of a Markov Chain and a vector of expected rewards (which, for a given action, depends only on the current state). A classic problem in MDPs, sometimes referred to as solving the MDP, is the problem of finding an optimal policy – i.e. a mapping from states to action such that if we choose the designated action in each state and follow the dynamics of the corresponding Markov chain, we will maximize the total discounted reward, i.e. the sum of all rewards obtained along the way, where the reward at step t into the future is multiplied (discounted) by γ^t, where the discount factor $0 < \gamma < 1$ is a constant known in advance. It is well-known that an optimal policy can be found that is deterministic, i.e. it always chooses the same action in a given state. At the time of this writing, the MDP problem is exactly solvable in practice for small instances. There are classic algorithms for computing the optimum policy that have been known for decades such as the Value Iteration algorithm, the Policy Iteration Algorithm and a reduction to an instance of Linear Programming. We will discuss in the next section why these approaches cannot be directly applied in modern practical applications with massive streams of data. Finally, a yet more general class of problems is the

Partially Observable Markov Decision Process, where we assume that there is an underlying MDP, but where we do not observe the state directly, but rather only by means of a known set of observations. Unfortunately, even in the case where we consider finitely many steps ahead, the problem is PSPACE-hard in general [3]. Hence the only viable approach to the POMDP problem is either to consider only a class of instances of the problem which has some special structure or to provide only an approximate solution. Due to these constraints, the POMDP formalism is currently less popular in practical applications. There is also the philosophical argument that we should focus on understanding the simpler system (MDP) first, before moving on to more complex ones such as the POMDP. In fact, many approaches to the POMDP problem either represent the POMDP as a continuous MDP over the belief space or a countable MDP over histories (the former is intractable because the belief space is continuous and the latter because there are prohibitively many histories; algorithms typically have to make some approximation to provide a solution in a reasonable time).

Therefore this work focuses on the MDP problem, being complex enough to model interesting features of reality and simple enough to be more likely to be tractable. The problem with traditional approaches to solving MDPs is that these algorithms require the full description of the system as input. This makes their verbatim versions completely impractical for systems with many states, for reasons both of computational complexity (polynomial in the size of the system) and statistical efficiency (experimentally estimating the transition matrix of a Markov chain with n states and no particular structure means there are $n \times (n-1)$ parameters, which is far too many for realistic n). Hence the need for state abstraction, which is a way of exploiting structure in the state space to obtain a description of the set of states which is richer than just enumerating all possible configurations. Combining the algorithms for solving MDPs with state abstraction is currently a very active field – there exist working algorithms that are beginning to be employed in the business world, but they typically do not have the properties that one may wish, such as a guarantee that they will not diverge or a guarantee that the solution computed with the state abstraction is in some sense close to the true optimum solution.

Although this work is not directly about applications, it would not be complete without at least outlining the possibilities. It is widely believed that the first publicly-known large-scale deployments of Reinforcement Learning will be in the area of advertising and consumer interaction; potentially later spreading to other business processes as well. It is also very likely that the autonomous vehicle industry (land, air and water-based) will adopt some techniques at some point, although it is difficult to determine how exactly since the commercial work in this area is proprietary. It is very likely that applications in other areas of robotics will follow. In finance, approaches to time-series analysis related to RL techniques have been used for years; it is very likely that more of RL is being used now. There are also promising applications in healthcare and epidemiology although using them in practice would require a serious discussion about moral and legal pre-requisites.

The problem of state abstraction, sometimes referred to as feature induction is not unique to MDPs or RL, but is a common theme in all of Machine Learning, and in a broader sense of all science (finding a right set of parameters to describe useful features of a system is usually the first step in producing models for that system). There are many approaches, ranging from theories such as RKHS through complex architectures like neural networks and the approximation of manifolds. For this work, we decided not to use these complex ideas and use a very simple linear model instead, where compression is achieved via simple matrix multiplication. The reason for that is that even in this simple case, it is not trivial whether algorithms will work or how they will behave. We believe that in applied mathematics, it is better to try to fully understand the simple foundational ideas first before moving on to more complex ones. We do appreciate this is not the only approach and indeed there have been instances where non-linear approaches have been put to practical use very impressively. Ultimately, we hope that some of the understanding that we are trying to establish for the linear case will eventually benefit the general, non-linear case as well – however, that is not the subject of this thesis. In chapter 1, section 1.4.2 of this thesis, we will see that the linear framework we consider captures the three desiderata that is it reasonable to put forward for features in Reinforcement Learning. First, the features have to

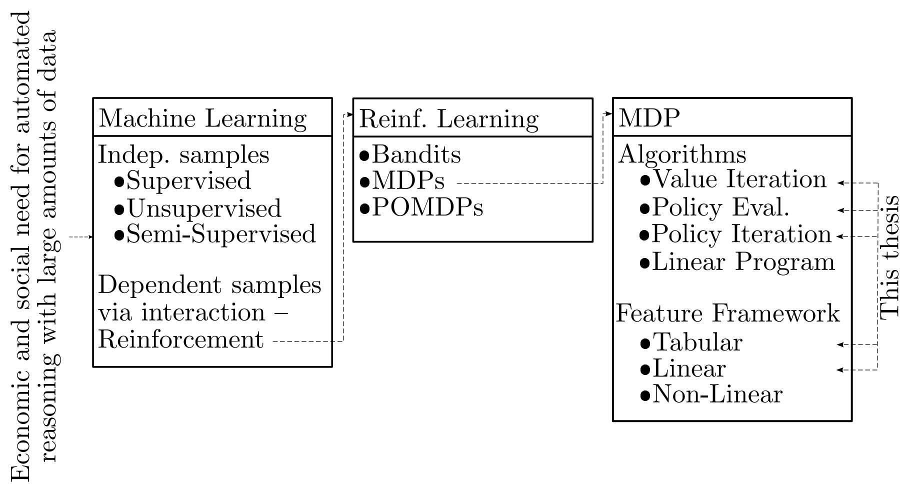


Figure 1: Taxonomy of ML algorithms.

model the value function well, i.e. they have to provide a good model to distinguish attractive parts of the state space from unattractive ones. Second, since Reinforcement Learning involves considering the effects of actions into the future, they have to model the dynamics of the system. Finally, the features should be compatible with the particular algorithm used to solve the MDP, a notion that will be clarified in section 1.4.1. Within the linear compression framework for MDPs, we discuss two problems, which we consider as useful and conceptually simple: evaluating a given policy and solving the MDP using value iteration and policy iteration. Variants of these problem occur very often as sub-cases in most practical applications of MDPs. Although there exist approaches where none of them is used, we argue that the two problems are canonical in that they occur frequently enough. Figure 1 summarizes the sections above by positioning the contents of this thesis in terms of the field of machine learning.

Another important issue that arises in practical approaches to solving MDPs is the question of how many steps ahead one should consider at a given iteration of the algorithm. Traditional algorithms always use just one time step, which may by suboptimal in that it leads to slow convergence for certain methods. It is believed that this can be improved [4] by using temporal abstraction – i.e. working with algorithms which consider the effects of a whole sequence of steps, rather than just one at a time. This idea is not new, and has been used in the planning community for a long time, typically in the form of macro-operators, sometimes arranged in a hierarchy. However, in Reinforcement Learning, a particularly elegant way of expressing non-deterministic operators can be defined, based on what is called options.[1] The assumption behind options is that we can define useful temporal abstractions by specifying two things: the behaviour at each state and the termination condition. In our framework, these can be inferred automatically by specifying the desired outcome that the option should have. This is done by defining a value function over states, called the sub-goal (if the sub-goal takes only two values: zero and a large constant, this boils down to just defining a subset of states). It is an open question whether this approach is useful in the sense that meaningful sub-goals can be found for practical problems – currently this requires substantial knowledge of the problem domain. Typically, reasonable sub-goals are easy to define but they may not lead to a practical speed-up of the MDP solution. In future it may be possible to efficiently construct sub-goals automatically using just the structure of the problem (current approaches to this exist, but have serious practical limitations). That being said, we decided it is an approach worth exploring since it is conceptually very simple and clear-cut. In our framework, a computed option model can be represented as a matrix and used in algorithms just like an ordinary one-step action, while the evaluation of an option with respect to a sub-goal is just a multiplication of a matrix times a vector. We do not consider our work a complete solution to the problem of temporal abstraction, but rather a useful early step.

To summarize, the overarching idea of this thesis is the study of feature-based Reinforcement Learning in a way which makes the most use possible of linear operators and their properties. This is motivated by the observation that even this case is not entirely trivial and leads to ideas that are likely to be useful in practice. The main concrete contributions are the following. First, in chapter 1 we use the concept of the joint spectral radius to develop a condition for when Reinforcement Learning with compressed models is stable in a certain sense. We also give a formula that quantifies the situation where a linear compression framework is optimal, which turns out to be an instance of the algebraic Riccati equation. We believe it is the first time the joint spectral radius and quadratic matrix equations

[1]This is entirely distinct from the meaning the word ‘option’ has in finance.

are applied to the analysis of approximate MDP solvers. Moreover, we describe a span of well-known Reinforcement Learning algorithms in a common matrix framework. Second, in chapter 2 we provide an in-depth analysis of the Least Squares Temporal Differences (LSTD) algorithm, and the comparison to Bellman Residual Minimization (BRM). In particular, we provide what we believe is the first correct derivation of LSTD by means of instrumental variables. We also give geometric analogies, expanding previous work. Third, in chapter 3 we formally introduce the concept of options in the context of Bellman operators defined on matrix models, again for the first time. Finally, we formally combine the linear framework with options, formulating a modified Value Iteration algorithm. We also describe empirical evidence from a simulated experiment. To our knowledge, this is the first time where an algorithm using options and value iteration efficiently solves medium-sized MDPs (our 8-puzzle domain has 181441 states). We demonstrate a modest improvement in runtime performance as well as a significant reduction in the number of iterations. Also, we have the first *convergent* VI-style algorithm where options (temporal abstraction) are combined with a framework for state abstraction, yielding far better results than the use of either idea alone.

Chapter 1

Linear Models

In this chapter, we will introduce concepts from Reinforcement Learning in the language of Linear Algebra. The concepts are not new, but it is necessary to provide exact definitions since we will employ them later throughout the thesis.

1.1 Classical Markov Models for Control

Many real-world control problems, can be modelled using the Markov assumption, where we identify a finite set of discrete system states in such a way that transitions from state to state can be described as depending on the previous state only. This framework is a generalization of planning in that transitions can be stochastic. We begin by defining the Markov Reward Process and the Markov Decision Process, which are the formalisms used to study such systems.

1.1.1 The Markov Reward Process

In this thesis we will consider systems with finite state-spaces of size n. It is well known that any finite-state Markov chain can be represented as a stochastic matrix (when we say stochastic, we mean right-stochastic, i.e. a matrix with real non-negative entries such that the rows sum to one). We now begin by introducing the Markov Reward Process, or MRP. It consists of two parts: the stochastic matrix P that describes state transitions (i.e. P_{ij} is the probability of transitioning from state i to state j) and the reward vector R which describes a reward attached to each state. The rewards either are deterministic functions of the state, or they are sampled from some probability distribution conditioned on the state. If the latter is the case, then the vector R contains the means of the distributions (i.e.

we have $R_i = \mathrm{E}[R \mid S = i])$ – for our purposes, this is indistinguishable from the rewards being deterministic.

Studying MRPs often involves working with *value functions*, or real functions defined over the state space. We will represent them as column vectors. Given an MRP, we now define the evaluation function associated with it as the discounted total sum of rewards we will obtain when we start in a given state and follow the dynamics. The discounting is by a factor $0 < \gamma < 1$, which guarantees that the function is well-defined.

$$V_A \equiv \sum_{t=0}^{\infty} (\gamma P)^t R = (I - \gamma P)^{-1} R$$

In the above, the second equality follows from the well-known von Neumann telescoping sum argument. The series converges and the inverse is guaranteed to exist because the eigenvalues of P are, as it is a stochastic matrix, guaranteed to be in the complex unit circle.

We will now introduce Sutton's homogeneous notation [5] to describe an MRP using just a single matrix. Indeed, consider the following block matrix.

$$A \equiv \left[\begin{array}{c|c} 1 & 0_{\square} \\ \hline R & \gamma P \end{array}\right] \tag{1.1}$$

This leads to the following formula for V.

$$A^{\infty} = \left[\begin{array}{c|c} 1 & 0_{\square} \\ \hline (I - \gamma P)^{-1} R & 0_{\square} \end{array}\right]; \quad \left[\begin{array}{c} 1 \\ \hline V_A \end{array}\right] = A^{\infty} \left[\begin{array}{c} 1 \\ \hline 0_{\square} \end{array}\right]$$

Here, by A^{∞} we denote the limit of $\lim_{t\to\infty} A^t$, which is guaranteed to exist because the eigenvalues of A consist of the eigenvalue one and the eigenvalues of γP. By $0_{\square}$ we denote the zero matrix or vector of the appropriate size.

We will now describe the properties of MRPs that have the matrix format of equation 1.1. Consider the situation where we have two MRPs A_1,A_2 defined over the same state space. It is clear from the definition of matrix multiplication that $A_1 A_2$ also has the required block format. Indeed, it is easy to verify that it is an MRP equivalent to first making a step according to A_1 and then according to A_2. We note here that although matrices having the format of equation 1.1 are closed under multiplication, they are not a group. We are now going to see that such matrices also can be thought as linear operators.

First, we will define rasps [6]. Rasps are row vectors of the form $\left[\begin{array}{c|c} z & \xi \end{array}\right]$, where z is some real number corresponding to accumulated discounted reward and ξ is a row vector defining a probability distribution. We see that MRPs in the matrix format are transformations from rasps to rasps. Indeed the multiplication $\left[\begin{array}{c|c} z & \xi \end{array}\right] A$, where A is an MRP in the matrix format has the following meaning: we start having accumulated z units of discounted reward, and our chances of being in a given state of the MRP are given by the distribution ξ. The multiplication than models making move according to the MRP, to obtain a rasp with a new reward and a new distribution of states. We note that the multiplication by a rasp is on the left of the MRP model.

There is also another linear operation we may do with matrix MRP models, where the multiplication is on the right. It corresponds to evaluating an MRP with respect to a given value function. Indeed, consider the following multiplication for some arbitrary value function V.

$$\left[\begin{array}{c} 1 \\ \hline V' \end{array}\right] = A \left[\begin{array}{c} 1 \\ \hline V \end{array}\right] \tag{1.2}$$

Note that the result also corresponds to some value function V'. Indeed, V' is the function obtained by, for each state, making one move according to A and then evaluating the state distribution we find ourselves in using the value function V (this is often referred to as the *action-value function*).

1.1.2 The Markov Decision Process

We will now define Markov Decision Processes (MDPs). An MDP is a finite set of MRPs $\{A_1, A_2, \ldots, A_l\}$ defined over the same state space. The MRPs are indexed with numbers $1, 2, \ldots, l$ and represent different actions. When we work with MDPs, it is often necessary to use the notion of a deterministic policy. A deterministic policy $\pi : \{1, \ldots, n\} \to \{1, \ldots, l\}$ is a function of the state which returns the index of some action. A policy can be represented as the MRP model M_π where the row corresponding to state j is taken from A_j in the following way.

$$\left[\begin{array}{c|c} 1 & e_i^\top \end{array}\right] A_\pi = \left[\begin{array}{c|c} 1 & e_i^\top \end{array}\right] A_{\pi(i)} \tag{1.3}$$

We denote the function that evaluates this model $V_\pi \equiv V_{A_\pi}$ and we call the process of computing it Policy Evaluation (PE). Here e_i is the column i of the identity matrix.

A common theme in the theory of MDPs is the problem of finding an optimal policy $\pi^\star$. It is defined as a policy that maximizes $\sum_i V_\pi(i)$. We sometimes slightly abuse the notation and refer to 'the' optimum policy when we mean any such maximizer because in applications, even if there are many valid choices, we do not typically care which one we pick. We could do this formally and work with equivalence classes of policies rather than policies themselves but we believe this would add no substance to our results and only obfuscate them. We call the evaluation of $\pi^\star$ the optimal value function and denote it $V^\star$.

In other work, it is frequently the case that one considers a generalization of the above concept, or stochastic policies. They are defined as a mapping form states to discrete probability distributions over actions, where the deterministic case is obtained when the whole probability weight is concentrated on one element. For classical MDPs as they are discussed in this section there is a well-known lemma – see theorem 6.2.7 of the monograph [7] – that states that the optimum value of $\sum_i V_\pi(i)$ over the general set of non-deterministic policies is attained for some deterministic policy. Therefore the simplest choice and the one we are making in our formal analysis here is to limit ourselves to deterministic policies. We note that there are other reasons why one might prefer non-deterministic policies in practice, such as the issue of exploration in sample-driven systems. If this is needed, then the generalization of our results to the non-deterministic case is trivial and consists in basically taking a weighted sum of rows from different action models, where the weights correspond to action probabilities.

1.1.3 Solving MDPs with Value Iteration

We will now define the classic algorithm used for solving MDPs in the language of matrix models we defined in the previous section. The algorithm is called Value Iteration [8] and consists in starting with an arbitrary initial value function and repeatedly applying to it a model corresponding to a policy which maximizes the value obtained by following the model and then evaluating using the existing value function (i.e. it always chooses the greedy action). Formally, we start with the initial value function V_0 (arbitrary, but in the absence of prior knowledge, this is typically assumed to be uniformly zero). We then apply the following formula to obtain V_t from V_{t-1} for $t = 1, 2, \ldots$. The formula is applied for each state i in the range $1, 2, \ldots, n$ (the formally inclined reader will accept that we abuse our notation slightly and identify states with their indexes – this could be formalized by introducing a bijection from the set of states to the index set but such a formalization would not add anything to our analysis).

$$\begin{aligned} V_t^{A_a} &= A_a \left[\begin{array}{c} 1 \\ \hline V_{t-1} \end{array} \right] \\ \left[\begin{array}{c} 1 \\ \hline V_t \end{array} \right] &= \operatorname{cmax} \left[\begin{array}{c|c|c|c} V_t^{A_1} & V_t^{A_2} & \ldots & V_t^{A_l} \end{array} \right] \end{aligned} \tag{1.4}$$

Here, the operator cmax denotes the maximum over column indexes, i.e. the maximum value in each row. Exactly the same algorithm can be also phrased in terms of multiplication of matrix models in the following way.

$$\begin{aligned} V_t^{A_a} &= A_a M_{t-1} \left[\begin{array}{c} 1 \\ \hline 0_\Box \end{array} \right] \\ \pi &= \operatorname{imax} \left[\begin{array}{c|c|c|c} V_t^{A_1} & V_t^{A_2} & \ldots & V_t^{A_l} \end{array} \right] \\ M_t &= A_\pi M_{t-1} \end{aligned} \tag{1.5}$$

The operator imax is similar to cmax but returns the *index* of the maximum element in a given row. These operators are unusual, but they will be very useful later on. The representation of the algorithm given by equation 1.5 is computationally inefficient but serves as a useful theoretical tool, which we will use later. The model M_1 that we start with is completely arbitrary, but it is convenient to assume that it has zeros in both the reward and the state transition parts.

We will now look at the convergence of the algorithm. It is well known that the sequence V_t converges. We will first outline the classic justification. This goes by the Banach contraction argument. First consider equation 1.4 as an operator equation, i.e. we define an operator $T^\star$, called the Bellman optimality operator which works on value functions such that $V_i = TV_{i-1}$ is by definition defined as in equation 1.4. It is well-known that this operator is a contraction in the norm L_∞. Note that the contraction property crucially depends on the transition matrix being stochastic. Therefore, by the Banach contraction theorem, it has a single fix-point which can be converged to by iterated applications of $T^\star$. Once we know that the sequence has converged it is easy to see by equation 1.4 that the limit equals $V^\star$ (see theorem 6.2.1 in the monograph by Puterman [7] for a formal argument). We note that the limit of the algorithm of equation 1.5 has the following form.

$$M_T^\infty \prod_{t=1}^{T-1} M_t \tag{1.6}$$

1.2 Compressed Markov Models

It is frequently the case that the number of states n in a system is so large that it is intractable to explicitly store model matrices which are that large. Hence there is the need for state abstraction. In this thesis, we are concerned with state abstraction of the linear form. More concretely, we are interested in reducing the complexity of working with an MRP by constructing a compressed version of size k where $k \ll n$. Denote the compression operators C (of size $k \times n$) and the decompression operator D (of size $n \times k$). We are making the central assumption that these operators are *linear* i.e. they are real matrices. We obtain the compressed reward vector as $\tilde{R} = CR$ and the compressed transition dynamics as $\tilde{P} = CPD$. In our matrix form, if the MRP is given by a matrix A, the compressed version $\tilde{A}$ is given by the formula below.

$$\underbrace{\left[\begin{array}{c|c} 1 & 0_\square \\ \hline \tilde{R} & \tilde{P} \end{array}\right]}_{\tilde{A}\in\mathbb{R}^{k\times k}} = \left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & C \end{array}\right] \underbrace{\left[\begin{array}{c|c} 1 & 0_\square \\ \hline R & P \end{array}\right]}_{A\in\mathbb{R}^{n\times n}} \left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & D \end{array}\right] \tag{1.7}$$

We assume that D is of full column rank. We stress that in this work, when we want to compress a whole MDP, we apply *the same* matrices C and D to all actions, which we will see later is important to assure convergence. This framework is a generalization of [9, 10] because we do not require C and D to be stochastic matrices or averagers. While computing good values for C and D is an unsolved problem in general, we will describe one interesting choice good choice of C and D in section 1.4.2.

The framework of equation 1.7 relates to existing approaches as follows: we obtain the baseline case i.e. table-lookup case when there is no compression, i.e. $D = C = I$. In this case, all matrices P are contractions in the sup-norm. A simple, more general way [10] of ensuring CPD to be a contraction is to make D and C rectangular stochastic matrices.[1] In that case, it follows that the matrix $\tilde{P}$ is stochastic as well.

1.2.1 Value Iteration with compressed models

The main problem with using the compression framework 1.7 is of course the fact that the matrices generated by equation 1.7 are no longer models of the form 1.1, because the matrix $\tilde{P}$ is no longer a stochastic matrix. Therefore, we will need to describe additional conditions necessary to ensure that solution algorithms converge. The most canonical algorithm is Value Iteration, which we already described and where the resulting model at time T is of the following form.

$$\prod_{t=1}^{T} \tilde{M}_t = \prod_{t=1}^{T-1} \left[\begin{array}{c|c} 1 & 0_\square \\ \hline \tilde{R}_{\pi_t} & \gamma\tilde{P}_{\pi_t} \end{array}\right] \tag{1.8}$$

[1]In this thesis we call any non-negative real matrix with rows summing to one stochastic, even if it is not square. Non-square matrices do not correspond to any Markov chain, but they are still useful.

In the above, it is not certain whether the greedy operation 1.5 will finally settle on one policy. By $\tilde{R}_{\pi_i}$ and $\tilde{P}_{\pi_i}$ we mean, for $i = 1, 2, \ldots, T$, the respective vectors and matrices obtained by extracting rows from the actions of the compressed MDP as follows, where A_{π_i} is defined as in equation 1.3.

$$\left[\begin{array}{c|c} 1 & 0_{\square} \\ \hline \tilde{R}_{\pi_i} & \gamma \tilde{P}_{\pi_i} \end{array}\right] \equiv \tilde{A}_{\pi_i}$$

In section 1.3, we are going to analyse when this algorithm indeed converges.

1.3 General Analysis of Convergence

We will now recall a fact from linear algebra that describes the condition that a set of matrices must fulfil so that any infinite product of matrices from the set converges. Consider a finite set of square matrices of the same size $\mathbb{B} = \{B_1, B_2, \ldots, B_m\}$. Consider $\mathbb{B}^t$ to be the set of all products of matrices from $\mathbb{B}$ of length t. The following quantity, called *the joint spectral radius* of $\mathbb{B}$, is very useful [11].

$$\begin{aligned} \rho(\mathbb{B}) &\equiv \limsup_{t\to\infty} \left\{ \|M\|^{1/t} : M \in \mathbb{B}^t \right\} \\ &\equiv \limsup_{t\to\infty} \left\{ \rho(M)^{1/t} : M \in \mathbb{B}^t \right\} \end{aligned} \tag{1.9}$$

The formal proof of the equivalence of the two limits can be found in literature [11]. Intuitively, while the value of the (classic) spectral radius $\rho(M)$ tells us how the norm of M^t grows with t, the value of the joint spectral radius $\rho(\mathbb{B})^t$ tells us how the norm of the product grows as we increase the number of matrices we multiply together. The norm in the expression above can be any sub-multiplicative matrix norm. An important inequality for bounding the joint spectral radius is the following result.

$$\rho(\mathbb{B}) \leq \max_i \{\|B_i\| : B_i \in \mathbb{B}\} \tag{1.10}$$

We stress that the matrix norm in this expression is arbitrary but fixed, i.e. we have to apply the same norm to all elements of $\mathbb{B}$. This is essentially a restatement of the classic contraction argument in terms of linear algebra: we want all our matrices to be contractions in some given norm. It should be noted that this is not the only bound one can have; the theory of the joint spectral radius also provides more elaborate tools, such as using (possibly piece-wise) Lyapunov functions which are not norms.

It is clear that if $\rho(\mathbb{B}) < 1$, then any infinite product of models from the set converges to zero. On the other hand, if $\rho(\mathbb{B}) > 1$, then by definition 1.9 there exists a sequence of matrices from the set such that the norm of the product becomes arbitrarily large as we multiply more and more matrices, i.e. the product diverges.

Consider now the set of models corresponding to all possible policies, i.e. $\mathbb{M} = \{\tilde{A}_\pi : \pi = \pi_1, \pi_2, \ldots, \pi_{l^k}\}$, where $\tilde{A}_\pi$ is defined as in equation 1.3, but using the compressed actions $\tilde{A}_i$ instead of the uncompressed actions A_i. One can immediately see that the set of eigenvalues of $\tilde{A}_\pi$ consists of the eigenvalues of $\tilde{P}_\pi$ and one. We will call an algorithm *weakly stable* if the models being iterated are always contained within some ball of constant radius (i.e. independent of the trajectory length). The following condition is sufficient to guarantee weak stability.

$$\rho(\mathbb{P}) \leq 1 \text{ where } \tilde{\mathbb{P}} \equiv \{\tilde{P}_\pi : \pi = \pi_1, \pi_2, \ldots, \pi_{l^k}\} \tag{1.11}$$

Here, we use a soft inequality instead of a sharp one because the matrix $\tilde{P}$ will be multiplied by γ, which is guaranteed by assumption to be less than one. We note that the above condition is stronger than we require in that it guarantees weak stability for *all* possible ways in which the policies can be sequenced, while we only really care about the orderings produced by the rules governing our algorithms. There are two reasons why we are using this stronger condition: simplicity and the fact that it allows us to define stability only in terms of the matrices $\tilde{P}$, in a way which automatically holds for all rewards. Intuitively, one has to consider all orderings to encompass all possible rewards since it is always possible to artificially construct a reward vector that will lead to selecting a particular policy. Furthermore, we suspect that reward-universal algorithms are more robust when sampling.

There is one more aspect that we have to consider: when $\rho(\mathbb{B}) = 1$ then an infinite product of matrices from the set may, in general, either converge or there may be oscillatory behaviour where the sequence does not converge but the norm is bounded. It is currently unknown whether a system satisfying 1.11 is convergent or only weakly stable. However, we discuss in section 1.3.1 that we can guarantee convergence under the additional assumption that the transition matrices are non-negative.

We note that our reasoning about stability extends to the case where we have convex combinations of models, such as is the case with λ-type algorithms. This is the case because joint spectral radius of a set is closed with respect to the convex hull [11].

1.3.1 Convergence with non-negative transition matrices

In the following lemma, we will show that the value iteration algorithm converges when we can guarantee that, in addition to criterion 1.11, the transition matrices and the reward vector have only non-negative elements.

Lemma 1. *Assume all compressed action matrices $\tilde{A}_i$, $i = 1, \ldots, l$ have non-negative entries. Assume further that equation 1.11 holds. Then the algorithm of equation 1.8, starting with the zero value function, converges.*

Proof. We will show that the sequence of value functions generated by the algorithm is non-decreasing. Because we know that it is bounded from criterion 1.11, this will guarantee convergence.

For any two models $\tilde{A}_1, \tilde{A}_2$ appearing subsequently in our algorithm, we have the following.

$$\tilde{A}_1 \left[\begin{array}{c} 1 \\ \hline 0_{\square} \end{array}\right] \leq \tilde{A}_1 \tilde{A}_1 \left[\begin{array}{c} 1 \\ \hline 0_{\square} \end{array}\right] \leq \tilde{A}_2 \tilde{A}_1 \left[\begin{array}{c} 1 \\ \hline 0_{\square} \end{array}\right]$$

Here, the first inequality follows because we have non-negative rewards and transition operators and hence $\tilde{R}_1 \leq \tilde{R}_1 + \tilde{P}_1 \tilde{R}_1$. The second inequality holds because at each step of the algorithm, the maximization operator chooses the model maximizing the value function at each step. $\square$

1.3.2 A property of non-negative transition matrices

We will describe one particularly nice property of non-negative transition matrices. We have seen that in order to satisfy condition 1.11, we need to bound by one the joint spectral radius of the whole set of matrices corresponding to all policies, i.e. all matrices that have corresponding rows taken arbitrarily from $\tilde{P}_1, \tilde{P}_2, \ldots, \tilde{P}_l$. We will see in the lemma below that for non-negative matrices, it is sufficient to fulfil a much simpler condition, namely, that the joint spectral radius of only the set $\tilde{P}_1, \tilde{P}_2, \ldots, \tilde{P}_l$ (i.e. the compressed transition matrices of each action) is bounded by one. This is important because there are many fewer actions than policies, and because manually analysing the properties of matrices that have rows taken from a given selection is hard.

Lemma 2. *Consider the set of compressed actions $\tilde{P}_1, \tilde{P}_2, \ldots, \tilde{P}_l$. Assume that they are element-wise non-negative. Then we have the following.*

$$\rho(\{\tilde{P}_\pi : \pi = \pi_1, \pi_2, \ldots, \pi_{l^k}\}) \leq \rho(\{\tilde{P}_1, \tilde{P}_2, \ldots, \tilde{P}_l\})$$

Here, the lemma is stated without proof (it follows from theorem 2 of [12]). The basic idea of the work [12] is this: for nonnegative matrices, the joint spectral radius of a set of matrices with rows taken from a certain set is identical to a quantity called the joint row radius, which can be be computed using only a small number of matrices containing the rows used to construct all the others.

1.4 Picking Good Features

In this section, we discuss a possible choice of the compression framework, defined by the matrices C and D. We do this by identifying useful properties that the framework should have and identifying the class of choices that has them. We begin by stating an intuitively desirable property.

$$CD = I \tag{1.12}$$

Intuitively it means that that the compression does not lose information in the compressed function, i.e. if we take a compressed value function, decompress it and compress again, we will get the same compressed value function we started with.

1.4.1 The argument for non-negative features

During the iteration of the algorithm, we need to ensure that the actions chosen in the approximate framework are in some way similar to the actions that would be chosen by the algorithm in the table-lookup case. Consider the update rule, which is the same as equation 1.4 except we use the compressed (tilde) models.

$$\tilde{V}_t^{\tilde{A}_a} = \tilde{A}_a \begin{bmatrix} 1 \\ \hline \tilde{V}_{t-1} \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ \hline \tilde{V}_t \end{bmatrix} = \operatorname{cmax} \left[\; \tilde{V}_t^{\tilde{A}_1} \;\middle|\; \tilde{V}_t^{\tilde{A}_2} \;\middle|\; \ldots \;\middle|\; \tilde{V}_t^{\tilde{A}_l} \; \right]$$

Now consider 'expanded' value functions of the form $D\tilde{V}_t^{\tilde{A}_i}$. Ideally, we would like the 'expanded' value function and the 'compressed' value function $\tilde{V}_t$ (note that $\tilde{V}_t = CD\tilde{V}_t$ due to equation 1.12) to lead to choosing equivalent actions, that is, we postulate that the following might naively be expected to hold.

$$\begin{aligned} C \operatorname{cmax} \left[\; D\tilde{V}_t^{\tilde{A}_1} \;\middle|\; D\tilde{V}_t^{\tilde{A}_2} \;\middle|\; \ldots \;\middle|\; D\tilde{V}_t^{\tilde{A}_l} \; \right] &\overset{\text{should}}{=} \\ \overset{\text{should}}{=} \operatorname{cmax} \left[\; CD\tilde{V}_t^{\tilde{A}_1} \;\middle|\; CD\tilde{V}_t^{\tilde{A}_2} \;\middle|\; \ldots \;\middle|\; CD\tilde{V}_t^{\tilde{A}_l} \; \right] \end{aligned} \tag{1.13}$$

Unfortunately, the only way of satisfying equation 1.13 for all value functions is hard aggregation – i.e. when the matrix C is binary and has exactly one 1 in each row. Here, the operator cmax again returns the maximum of the values in a given matrix row. Note that for non-negative entries, this is the same as the row-wise L_∞, but we do not restrict ourselves to non-negative value functions.

Since we cannot have equation 1.13, and we want to use features more expressive then just hard aggregation, we propose to instead use non-negative D for the following reason. Assume for the moment that the functions $\tilde{V}_t^{A_a}$ are constant, i.e. all rows of the matrix in equation 1.13 are the same. We see that in this case, if we have non-negative D then we will have equality, while if we have D that is all negative, the operator cmax on the right hand side will in fact choose the *worst* action. Note that the constraint to use non-negative features is not a problem in practice; the reasoning we are going to use in section 1.4.2 only depends on the subspace spanned by the features, not on its basis – we can therefore take arbitrary features, find a non-negative basis which spans a subspace containing the original features, and use the non-negative features. We stress here that the requirement for non-negativity is clearly only necessary, but it is not sufficient to ensure a good approximation.

1.4.2 A characterization of good features in terms of P.

In this section we will describe an algebraic condition that can be used to obtain good features for the algorithm. We begin with a single MRP and will generalize our findings to the MDP case later in section 1.4.4. We are interested in evaluating the value function of the MRP given by $V = (I - \gamma P)^{-1} R$ using the compressed version of the MRP. We approximate the value function by the span of the matrix D, i.e. the approximate value function is of the form $D\tilde{V}$. In particular, we will call an approximate value function $D\tilde{V}$ *transition-optimal* if $D\tilde{V} = \Pi V$. Here, we define Π as follows.

$$\Pi = DC$$

For now Π may not necessarily be a projection (but we will see later that it is useful to limit Π to be a projection – see observation 1). We note here that according to the definition above, a transition-optimal approximate value function can still be useless in practice, because ΠV can be very different from V and can lead to a completely different policy. Hence in practice, we will need two conditions: optimality and the fact that ΠV is close to V in some sense.

We want to describe the class of matrices Π satisfying the condition that the approximation is transition-optimal for the MRP. $\tilde{V}$ is the value function of the compressed MRP, i.e. $\tilde{V} = (I - \gamma \tilde{P})^{-1} \tilde{R} = (I - \gamma CPD)^{-1} CR$. We have the following equation for optimality.

$$D\tilde{V} = \Pi V \;\Leftrightarrow\; D(I - \gamma CPD)^{-1} CR = DC(I - \gamma P)^{-1} R$$

We assume this has to hold for all choices of R and we also assume that the columns of D are linearly independent. This gives the condition $(I - \gamma CPD)^{-1}C = C(I - \gamma P)^{-1}$, which simplifies to $CP = CPDC$, which holds if and only if the following condition for Π holds.

$$\Pi P = \Pi P \Pi \tag{1.14}$$

Hence we have described the family of matrices Π satisfying the optimality condition. Note that equation 1.14 is a special case of the algebraic Riccati equation [13]. Intuitively, equation 1.14 guarantees that our compression framework models the transition dynamics correctly. We will now describe a possible solution of 1.14. It is more convenient to work with the transposed version.

Observation 1. *When the matrix Π satisfies $\Pi^\top P^\top = P^\top$, it is a solution of the equation $\Pi P = \Pi P \Pi$.*

Proof. $\Pi P = \Pi P \Pi \;\Leftrightarrow\; P^\top \Pi^\top = \Pi^\top P^\top \Pi^\top \;\Leftrightarrow\; (\Pi^\top P^\top - P^\top)\Pi^\top = 0_\square$ □

We will now focus on finding a suitable Π which works for the particular transition matrix P that we are given. Because the row space of P has an intuitive interpretation, a robust approach to do that is to simply build up a matrix Φ, which spans a space containing the row space of P. Now, we have the following, for some B such that the inverse $(\Phi^\top B)^{-1}$ exists.

$$\Pi = DC = \underbrace{B}_{D} \underbrace{(\Phi^\top B)^{-1}\Phi^\top}_{C} \tag{1.15}$$

Now, the simplest choice is to choose $B = \Phi$, so that we have an orthogonal projection and we have $C = \Phi^\dagger$ and $D = \Phi$, where by $(\cdot)^\dagger$ we denote the Moore-Penrose pseudo-inverse. In this case, we should choose Φ to consist of two sets of columns, the first being the basis of the row space of P and the second being some set of vectors which we suppose will model the value function well. This is intuitively easy to understand: in RL we need our features to model *three* things well: the dynamics of the system (represented by equation 1.14), the optimum value function and the action selection condition of section 1.4.1. The benefit of this approach is that this requirement is made explicit – in practical implementation it is easy to focus exclusively on modelling value functions and forget about the dynamics.

We will see in the following observation that any Π such that $\Pi^\top$ is a (possibly oblique) projection on an invariant subspace of $P^\top$ satisfies equation 1.14.

Observation 2. *Any matrix Π of the form $B(\Phi^\top B)^{-1}\Phi^\top$ where there exists some E such that $P^\top \Phi = \Phi E$ (i.e. the columns of Φ correspond to invariant subspaces of $P^\top$) satisfies the equation $\Pi P = \Pi P \Pi$.*

Proof. $\Pi P = \Pi P \Pi \;\Leftrightarrow\; P^\top \Pi^\top = \Pi^\top P^\top \Pi^\top \;\Leftrightarrow\; P^\top \Phi = \Pi^\top P^\top \Phi \;\Leftrightarrow\; \Phi E = \Pi^\top \Phi E \;\Leftrightarrow\; \Phi E = \Phi E$ □

We note that our work in this section is somewhat related to the invariant subspace method previously described in literature. In the work of [14] a choice of features based on invariant subspaces of P, was arrived at by first considering the Krylov basis, which is equivalent to the Bellman Error Basis Function (BEBF) basis and then setting the Bellman Error to zero, which is equivalent to $\Phi(\Phi^\top \Phi)^{-1}\Phi$ being a projection on an invariant subspace of P.

1.4.3 Perfect compositionality

We will now outline a connection between the Riccati equation we have introduced and a useful property of the compression framework, which we call the *perfect compositionality property.* This section contains[2] ideas introduced in the work of Singh and Sorg [15], who were the first to introduce *perfect compositionality* of linear models. The idea behind perfect compositionality is as follows: ideally, we would want the following. Consider some set of stochastic matrices of size $n \times n$, which we denote as $\mathbb{S}$.

$$\begin{aligned} &\forall P, P' \in \mathbb{S}.\; \Pi P P' = \Pi P \Pi P' \;\Rightarrow \\ &\forall P, P' \in \mathbb{S}.\; C P P' D = C P D \, C P' D \end{aligned} \tag{1.16}$$

[2]We developed this notion independently and only discovered the paper [15] afterwards. Also, our exposition is more general, because we consider all solutions of the Riccati equation, not just orthogonal projections. Also, we provide a direct link between the perfect compositionality property (which is equivalent to the set of Riccati equations 1.18) and convergence of model-based algorithms – see section 1.6.2.

Intuitively, this means that the compression of the composition is the same as the composition of compressed operators. We have the following lemma, which states that this condition is exactly equivalent to the Riccati Equation.

Lemma 3. *Assume $I \in \mathbb{S}$. The condition of equation 1.16 is equivalent to each member of the set $\mathbb{S}$ satisfying the Riccati equation (eq. 1.14) for a single* Π.

Proof. One direction of the implication follows by taking $P' = I$. The other direction follows by post-multiplying by P'. $\square$

1.4.4 The full MDP case

Given an MDP, our features need to solve the following *system* of Riccati equations.

$$\forall a \in 1, \ldots, l. \quad \Pi P_a = \Pi P_a \Pi \tag{1.17}$$

As with the case of a single equation, again the simplest thing we can do is to consider oblique projections along the space orthogonal to the concatenated row space of all matrices P_a, $a = 1, \ldots, l$. This allows us to construct the projection without knowing the eigenvector structure of the Ps. We can, if our transition matrices P are of low rank, simply use as our features some basis of the row-space of the vertically concatenated transition matrix for all the actions.

$$\begin{bmatrix} P_1 \\ P_2 \\ \vdots \\ P_l \end{bmatrix}$$

We note that if we do this we automatically obtain the following more general statement, which will be useful in section 1.7.

$$\forall \pi, \ \Pi P_\pi = \Pi P_\pi \Pi \tag{1.18}$$

1.5 Other RL Algorithms

For the case where the matrices $\tilde{P}$ are such that we can guarantee convergence, we present the most common algorithms used in Reinforcement Learning by putting models in our matrix form. The result of all the described algorithms will be an infinite multiplication of model matrices, where the next matrix is generated from the previous according to some rule. The most canonical algorithm is Value Iteration, which we already described and where the resulting model t time T is of the following form.

$$\prod_{t=1}^{T} \left[\begin{array}{c|c} 1 & 0_\square \\ \hline \tilde{R}_{\pi_t} & \gamma\tilde{P}_{\pi_t} \end{array}\right]$$

In the above, T is some time index at which the greedy operation 1.5 settles on one policy. By $\tilde{R}_{\pi_i}$ and $\tilde{P}_{\pi_i}$ we mean, for $i = 1, 2, \ldots, T$, the respective vectors and matrices obtained by extracting rows from the actions of the compressed MDP as follows, where A_{π_i} is defined as in equation 1.3.

$$\left[\begin{array}{c|c} 1 & 0_\square \\ \hline \tilde{R}_{\pi_i} & \gamma\tilde{P}_{\pi_i} \end{array}\right] \equiv \tilde{A}_{\pi_i}$$

Policy Iteration, on the other hand, is an algorithm where the rule defining the generation of one model from another is different, and given below.

$$\begin{aligned} \tilde{V}_t^{\tilde{A}_a} &= \tilde{A}_a \tilde{M}_{t-1} \left[\begin{array}{c} 1 \\ \hline 0_\square \end{array}\right] \\ \pi &= \operatorname{imax}\left[\begin{array}{c|c|c|c} \tilde{V}_t^{\tilde{A}_1} & \tilde{V}_t^{\tilde{A}_2} & \ldots & \tilde{V}_t^{\tilde{A}_l} \end{array}\right] \\ \tilde{M}_t &= (\tilde{A}_\pi)^\infty \tilde{M}_{t-1} \end{aligned} \tag{1.19}$$

In this case, the result of the algorithm at time T is a matrix product of the form

$$\prod_{t=1}^{T} \left[\begin{array}{c|c} 1 & 0_\square \\ \hline \tilde{R}_{\pi_t} & \gamma\tilde{P}_{\pi_t} \end{array}\right]^\infty$$

There is also an intermediate version called Modified Policy Iteration [16], where we start with a finite constant L, the generation rule is similar to the one used by Policy Iteration (we exponentiate to the power of L, not ∞) and the result at time T is of the following form.

$$\prod_{t=1}^{T} \left[\begin{array}{c|c} 1 & 0_{\square} \\ \hline \tilde{R}_{\pi_t} & \gamma \tilde{P}_{\pi_t} \end{array} \right]^{L}$$

It is straightforward to provide an extension to Modified Policy Iteration where L is not a constant but a parameter that changes, for instance in response to some property of the current policy or the current value function. One could also attempt to control L with some independent process, treating it as some sort of 'temperature'. However, we do not concern ourselves with such modifications in this thesis. We stress only that the convergence analysis is also valid for all such modifications.

Another possible modification is where the algorithm produces a product of matrices which are *convex combinations* of (possibly exponentiated) models corresponding to policies. One possible choice is λ-type algorithms [10], where the matrix product at time T is of the following form.

$$\prod_{t=1}^{T} \left(\sum_{p=0}^{\infty} \lambda^p (1-\lambda) \left[\begin{array}{c|c} 1 & 0_{\square} \\ \hline \tilde{R}_{\pi_t} & \gamma \tilde{P}_{\pi_t} \end{array} \right]^{p+1} \right)$$

We stress that from the point of view of convergence, it doesn't matter which convex combination we take. It is also possible to use a different combination at each step, tuning the parameters as the algorithm progresses.

1.6 Summary of results on convergence

1.6.1 Convergence in the non-negative case

We have seen in sections 1.3.2 and 1.3.1 that if we can guarantee the compressed actions are non-negative and fulfil the condition $\rho(\{\tilde{P}_1, \ldots, \tilde{P}_l\}) \leq 1$ then the value iteration algorithm converges. We will now discuss a condition for the matrices $\tilde{P}_1, \ldots, \tilde{P}_l$ to have non-negative entries. In particular, we have the following observation, which tells us that a matrix $\tilde{P}$ is non-negative if P can be factorized in a certain way.

Observation 3. *If the matrix P can be factorized as $P = \Phi E$, where Φ and E are matrices with non-negative entries, then the matrix $\tilde{P} = CPD$ has non-negative entries, where $C = \Phi^\dagger$ and $D = \Phi$,*

Proof. $\tilde{P} = CPD = \Phi^\dagger P \Phi = \Phi^\dagger \Phi E \Phi = E\Phi$ □

We note here that unfortunately this way of factoring the matrix P is contrasts with the Riccati equation of section 1.4.2 as follows: in section 1.4.2 it was convenient to pick a Φ which spans the *row* space of P, but here we need Φ to span for the *column* space of P.

1.6.2 Convergence with the Riccati equation

We have seen beforehand that the algorithms of section 1.5 are weakly stable (i.e. do not diverge without bound) if the condition 1.11 is satisfied. In particular, they are weakly stable if for any aggregate policy, the model $\tilde{A}_\pi$ is a contraction wrt. some vector norm independent of π. In this section, we show that if the compression framework satisfies the Riccati equation, we can make the stronger statement that the algorithms will converge to a single fixpoint.

Indeed, consider the case when we have that the system of Riccati equations $\Pi P_i = \Pi P_i \Pi$ holds for each action $i = 1, \ldots, l$. We can then show that this implies that each matrix $\tilde{P}_i$ is a non-expansion. Indeed, we have the following lemma.

Lemma 4. *For any $\Pi = DC$, if Π satisfies $\Pi P = \Pi P \Pi$, then $\tilde{P} = CPD$ is a non-expansion in some norm not depending on P.*

Proof. Before we begin, let us state that $\Pi P = \Pi P \Pi \Leftrightarrow \Pi^\top P^\top C^\top = P^\top C^\top$, where we use the fact that the matrix $D^\top$ has independent rows. Now, it is convenient to work with the transposed version. Consider the vector norm $\|C^\top \cdot\|_1$ We will show that $\tilde{P}^\top = (CPD)^\top$ is a non-expansion in this norm. Indeed, we have $\|C^\top D^\top P^\top C^\top x\|_1 = \|\Pi^\top P^\top C^\top x\|_1 = \|P^\top C^\top x\|_1 = \|C^\top x\|_1$. We thus have that $\tilde{P}^\top$ is a non-expansion in the norm $\|C^\top \cdot\|_1$. Note that $\|C^\top \cdot\|_1 = \|C^\top \Lambda^{-1} \Lambda \cdot\|_1 = \|\Lambda \cdot\|_1$, where Λ is a diagonal matrix containing on the main diagonal the L_1 norms of each column of the matrix $C^\top$. We therefore see that $\tilde{P}$ is a non-expansion in the norm $\|\Lambda^{-1} \cdot\|_\infty$. □

Observe that the norm $\|\Lambda^{-1} \cdot \|_\infty$ operates row-wise, i.e. it we only have to guarantee that all compressed actions are non-expansions, not all policies. Indeed, we are now going to show that when the assumptions of lemma 4 hold, the value iteration algorithm converges.

Lemma 5. *For any set of compressed actions with transition parts $\gamma\tilde{P}_i$, if each $\tilde{P}_i$ is a non-expansion in the norm $\|\Lambda^{-1} \cdot \|_\infty$ for some non-negative Λ^{-1} single for all actions, then the generalized Bellman optimality operator $\tilde{T}^*$, defined as $(\tilde{T}^* V)(i) = \max_a \tilde{R}_a(i) + \gamma\tilde{P}_a(i, :)V$, is a contraction.*

Proof. The proof goes through in a way very similar to the proof for the table-lookup case. We want to show the following.

$$\|\Lambda^{-1}(\tilde{T}^*\tilde{V}_1 - \tilde{T}^*\tilde{V}_2)\|_\infty \leq \gamma\|\Lambda^{-1}(\tilde{V}_1 - \tilde{V}_2)\|_\infty$$

Introduce π_1 and π_2 as the greedy policies wrt. $\tilde{V}_1$ and $\tilde{V}_2$ respectively. We want to find a policy π_3, such that we have the following.

$$\|\Lambda^{-1}(\tilde{T}^*\tilde{V}_1 - \tilde{T}^*\tilde{V}_2)\|_\infty = \|\Lambda^{-1}(\tilde{T}_{\pi_1}\tilde{V}_1 - \tilde{T}_{\pi_2}\tilde{V}_2)\|_\infty \leq \|\Lambda^{-1}(\tilde{T}_{\pi_3}\tilde{V}_1 - \tilde{T}_{\pi_3}\tilde{V}_2)\|_\infty$$

We can construct such a π_3 using row-by row argument, by setting $\pi_3(x) = \pi_1(x)$ if $(T_{\pi_2}\tilde{V}_2)(x) \leq (T_{\pi_1}\tilde{V}_1)(x)$ and else $\pi_3(x) = \pi_2(x)$. The only difference between this case and the table-lookup case is that we cancel the appropriate diagonal element of Λ^{-1} on both sides, i.e. we have that $|(\tilde{T}_{\pi_1}\tilde{V}_1 - \tilde{T}_{\pi_2}\tilde{V}_2)(x)| \leq |(\tilde{T}_{\pi_3}\tilde{V}_1 - \tilde{T}_{\pi_3}\tilde{V}_2)(x)|$ implies $|\Lambda^{-1}(x)(\tilde{T}_{\pi_1}\tilde{V}_1 - \tilde{T}_{\pi_2}\tilde{V}_2)(x)| \leq |\Lambda^{-1}(x)(\tilde{T}_{\pi_3}\tilde{V}_1 - \tilde{T}_{\pi_3}\tilde{V}_2)(x)|$. We now continue as follows: $\|\Lambda^{-1}(\tilde{T}_{\pi_3}\tilde{V}_1 - \tilde{T}_{\pi_3}\tilde{V}_2)\|_\infty \leq \gamma\|\Lambda^{-1}\tilde{P}_{\pi_3}(\tilde{V}_1 - \tilde{V}_2)\|_\infty \leq \gamma\|\Lambda^{-1}(\tilde{V}_1 - \tilde{V}_2)\|_\infty$. $\square$

1.7 Perfect Compositionality and Large Policies

We will now consider, as an aside, a changed version of the value iteration algorithm which impractical, but serves as a useful theoretical tool. Consider a value iteration algorithm of the following form.

$$M_t = \max_\pi 1_\square^\top (\widetilde{A_\pi}) M_{t-1} \left[\begin{array}{c} 1 \\ \hline 0_\square \end{array}\right] \quad \text{where} \tag{1.20}$$

$$(\widetilde{A_\pi}) = \left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & C \end{array}\right] \left[\begin{array}{c|c} 1 & 0_\square \\ \hline R & P_\pi \end{array}\right] \left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & D \end{array}\right]$$

We note a crucial difference from algorithm of section 1.2.1 – here, the policies are constructed not by taking rows out of compressed models, but by compressing the policies from the original MDP. This is impractical because there are l^n such policies, but the algorithm is interesting nonetheless.

We will now concern ourselves with the question of what the algorithm converges to, assuming we have the perfect compositionality introduced in section 1.4.3, i.e. we assume that equation 1.18 is fulfilled. To do that, we will consider the implications of the perfect compositionality property for model multiplication. We would like the following equality to hold. Intuitively it means that compression of a composition is a composition of compressions.

$$(\widetilde{A_1 A_2}) = \left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & C \end{array}\right] A_1 A_2 \left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & D \end{array}\right] = \tag{1.21}$$

$$= \underbrace{\left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & C \end{array}\right] A_1 \left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & D \end{array}\right]}_{\tilde{A}_1} \underbrace{\left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & C \end{array}\right] A_2 \left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & D \end{array}\right]}_{\tilde{A}_2}$$

Denote by P_1, P_2 and R_1, R_2 the transition and reward parts of A_1 and A_2. It is easy to see that the above equality holds provided the system 1.18 is satisfied.

We have seen in section 1.2.1 that what the algorithm converges to can be represented as a product of matrices. We can therefore apply equation 1.21 iteratively. The question is whether the algorithm converges to $\Pi V^\star$. Unfortunately, we can only assure this for the case when we have satisfied the condition 1.13, i.e. for hard aggregation. When this is not the case, the approximate algorithm may choose a different sequence of actions from the table-lookup one, hence converging to a different solution.

1.8 Summary of Contributions

We have presented a unified framework, which serves as a useful tool to express many common RL algorithms. We have introduced the joint spectral radius as a criterion of weak stability of these algorithms. We have also derived a condition for the optimality of features used in RL, which turns out to be an algebraic Riccati equation. We have formally shown that features used in a linear RL framework have to meet 3 criteria: they have to model the value functions and the transition dynamics well, and they have to be compatible with the algorithm in the way we described in section 1.4.1. Finally, for the case where the compressed transition matrix and the compressed expected reward vector are non-negative, we can show that the algorithms are convergent. Finally, we have shown that for a certain class of algorithms of section 1.7, the Riccati equation corresponds to the perfect compositionality property, which means that a compression of model combination is a combination of compressions.

Chapter 2

Policy Evaluation with Compressed Models

In this chapter, we present a very popular application of the ideas of linear modelling introduced in chapter 1, which is policy evaluation using the well-known Least Squares Temporal Differences (LSTD) algorithm. We define the algorithm as a way of evaluating an approximate model of a policy (a linear dynamical system). We also give alternative ways of looking at the algorithm: the operator-theory approach via the Galerkin method, the statistical approach via instrumental variables as well as the limit of the TD iteration. Further, we give a geometric view of the algorithm as an oblique projection. Moreover, we compare the optimization problem solved by LSTD as compared to Bellman Residual Minimization (BRM). We also treat the modification of LSTD for the case of episodic Markov Reward Processes.

The main practical problem that the LSTD algorithm solves is such: we are given a feed of data from a stochastic system, consisting of a state description in terms of features and of rewards. The task is to construct an abstraction that maps from states to values of states, where the value is defined as the discounted sum of future rewards. We will show that for LSTD, this abstraction is a linear model of the kind we introduced in chapter 1. For example, the system may describe a chess game, the features of state may describe what pieces the players have while the reward signal corresponds to wither winning or losing the game. The value signal will then correspond to the value of having each particular piece. Note that this is not a general constant but may depend on the way the individual players play the game, for example the values may be different for humans than for computer players. We have seen in the previous chapter that the value function of a given policy can be expressed as $V = (I-\gamma P)^{-1}R$. The LSTD algorithm can be thought as a way of computing the value of this function approximately. The motivation for why the approximation is often necessary is threefold. First, we may not have access to the states directly, just to functions ϕ of state. Second, the number of states n is often computationally intractable. Third, even if n is tractable, there is the problem of statistical tractability – the number of samples needed to accurately estimate transition matrices $n \times n$ is often completely prohibitive.

Associated with our problem setting is the question whether the value function is interesting in its own right, or whether we only need it to adjust the future behaviour of some aspect of the environment we can control (i.e. in our chess example make a move). We believe that there is large scope for systems (for instance expert systems) where the focus will be on gaining insight into the behaviour of the stochastic system, but the decisions about whether or how to act will still be made manually by human controllers, on the basis of the value-function information. These are the cases where algorithms like LSTD are the most directly applicable. On the other side of the spectrum, there will also of course be situations where the value function estimate is used as a tool to automatically generate the best action on the part of the agent – such systems may also use value-function estimation algorithms of the type of LSTD to operate within the policy iteration framework, or more generally, one of the algorithms introduced in chapter 1.

2.1 Prior Work on LSTD

An exhaustive introduction to least-squares methods for Reinforcement Learning is provided in chapter 6 of Bertsekas' monograph [10]. The LSTD algorithm was introduced in the paper by Bradtke and Barto [17]. Boyan later extended to the case with eligibility traces [18], wherean additional parameter λ controls how far back the updated are influenced by previous states. The connection between LSTD and LSPE, as well as a clean-cut proof that the on-line version of LSTD converges, was given by Nedić and Bertsekas [19]. The seminal paper [20] by Tsitsiklis and Van Roy provided an explicit connection between the fix-point of the iterative TD algorithm and the LSTD solution, while also formally proving that the TD algorithm for policy evaluation converges. The paper [21], described the Bellman Residual Minimization procedure as an alternative to TD. Antos' paper [22] provided an extensive comparison on the similarities and differences between LSTD and Bellman Residual Minimization (BRM). Parr's paper [23] introduced the LSPI algorithm as a principled way to combine LSTD with control. The paper by Munos [24] introduced bounds for policy iteration with linear function approximation, albeit under strong assumptions. Scherrer provided [25] the geometric interpretation of LSTD as an oblique projection, in the context analysing the differences between LSTD and BRM. The paper [26] represents an early approach to automatically constructing features for RL algorithms, including LSTD. Schoknecht gave [27] an interpretation of LSTD and other algorithms in terms of a projection with respect to a certain inner product. Choi and Van Roy [28] discuss the similarities between LSTD and a version of the Kalman filter. There exist various approaches in literature to how LSTD can be regularized, none of which can be conclusively claimed to outperform the others. These include the L1 approaches of [29] and [30] and the nested approach of [31]. These approaches differ not just in the what regularization term in used, but they solve different optimization problems (we will discuss this in section 2.5).

2.2 Definition of LSTD

2.2.1 Notation

The LSTD algorithm finds the value function of a finite-state Markov Reward Process (MRP), which we defined in the previous chapter. The MRP is fixed, i.e. we only consider the on-policy setting. However, while we previously assumed the knowledge of the full model, now we only have access to linear features of states and to the obtained rewards. More formally, denote as P the transition matrix of the MRP. For each state s we have a feature row-vector ϕ. The feature design matrix Φ gives the features of all states of the MRP, row-wise, where we assume that Φ has independent columns. We use the vector R, the i-th element of which contains mean reward obtained while leaving the state i. We use ξ to denote a left eigenvector of P corresponding to eigenvalue one. Note that if the chain has a stationary distribution, it will correspond to such an eigenvector, but we do not require it. We will assume that the chain only has one recurrent class since the case where we have many classes complicates the notation without contributing to the main argument (in practice, we can typically assume there is one class if we do enough exploration). We also introduce the matrix $\Xi = \mathrm{diag}(\xi)$. We now define expectations of functions of the Markov process in terms of weighted averages. For example the expectation of $\phi^\top \phi$, is defined by $\mathrm{E}\big[\phi^\top \phi\big] = \Phi^\top \Xi \Phi$, and similarly for other functions. By this we mean that if P is ergodic, it is legitimate to consider the above quantity an expectation corresponding to long-time average by the standard ergodic theorem for Markov chains. But in our application it is convenient to be more general and allow for periodicity, i.e. the diagonal of Ξ may not be a stationary distribution, but the expression still matches the long-time average. We use subscripts to denote two-step sampling, for example ϕ'_s denotes the fact that we first sample a state, then the successor state and obtain the feature of that successor state. When we write an expectation w.r.t. such a variable, for example $\mathrm{E}[r_s^2]$, the distribution we mean for r is $\sum_{s=1}^{S} p(r|s)\xi_s$. In contrast to the derivations of chapter 1, we we were only concerned with the means, part of our present derivations depends on treating some qualities as random variables; we use small letters to denote them, for instance s denotes state and ϕ denotes feature. Once we have obtained samples from our process, we store them in matrices $\hat{\Phi}$ and $\hat{r}$, whose i-th rows correspond to, respectively, the state feature vector and reward obtained at time i. Observe the difference between Φ and $\hat{\Phi}$ – in the first one, each state is represented once, in the second one the number of rows corresponds to the trajectory taken in the MRP and repetitions are possible. The value function is discounted with the factor $0 < \gamma < 1$. Moreover, we introduce the square matrix S which has ones on the main diagonal and $-\gamma$ on the diagonal above it. It is the sample based equivalent to the operator $I - \gamma P$.

$$S = \begin{bmatrix} 1 & -\gamma & & \\ & \ddots & \ddots & \\ & & 1 & -\gamma \\ & & & 1 \end{bmatrix}$$

2.2.2 The linear dynamical system approach

The derivation given in this section is based on [14]. We begin by constructing a MRP which lives in the space of features instead of our original state space. We limit ourselves to the class of linear dynamical systems introduced in chapter 1. We need to define the matrix $\tilde{P}$ and the vector $\tilde{R}$, so that a transition from ϕ to ϕ' (row vectors) is modelled by $\phi\tilde{P} = \phi'$, and the reward we expect at ϕ is modelled by $\phi\tilde{R} = r$. Now we look for the values for $\tilde{P}$ and $\tilde{R}$ that model our system dynamics. We have that $\Phi\tilde{P}$ should be approximately equal to $P\Phi$ and $\Phi\tilde{R}$ to r. We weight states by Ξ, giving the following optimization problems.

$$\begin{aligned}\tilde{P} &= \underset{\tilde{P}}{\operatorname{argmin}} \|\Phi\tilde{P} - P\Phi\|_{\Xi} = \underset{\tilde{P}}{\operatorname{argmin}} \operatorname{trace}\left((\Phi\tilde{P} - P\Phi)^{\top}\Xi(\Phi\tilde{P} - P\Phi)\right) \\ \tilde{R} &= \underset{\tilde{R}}{\operatorname{argmin}} \ \|\Phi\tilde{R} - R\|_{\Xi} = \underset{\tilde{R}}{\operatorname{argmin}} \ (\Phi\tilde{R} - R)^{\top}\Xi(\Phi\tilde{R} - R) \end{aligned} \tag{2.1}$$

These optimization problems correspond to ordinary least squares (generalized to matrices in case of $\tilde{P}$) and the solutions are obtained by weighted projection: $\Phi\tilde{P} = \Pi P\Phi$ and $\Phi\tilde{R} = \Pi R$, where the projection matrix is defined as $\Pi = \Phi(\Phi^{\top}\Xi\Phi)^{-1}\Phi^{\top}\Xi$ and the matrix Φ cancels with the one in the projection, since it is full column rank. Now consider a feature vector ϕ. In the new approximate MRP, we can compute the value function exactly (i.e. all the approximation has already taken place when we constructed the matrix $\tilde{P}$ and vector $\tilde{R}$). The true value function associated with it is the expected discounted future reward, and is expressed as follows.

$$\phi \underbrace{\sum_{i=0}^{\infty}(\gamma\tilde{P})^{i}\tilde{R}}_{\tilde{V}} = \phi \underbrace{(I - \gamma\tilde{P})^{-1}\tilde{R}}_{\tilde{V}} \tag{2.2}$$

In the above, the last equality is the well known von Neumann telescoping sum argument as introduced in chapter 1. We thus have the equation $(I - \gamma\tilde{P})\tilde{V} = \tilde{R}$. In the above, we assumed that the series $\sum_{i=0}^{\infty}(\gamma\tilde{P})^{i}\tilde{R}$ converges. We show a stronger condition, namely that the series $\sum_{i=0}^{\infty}(\gamma\tilde{P})^{i}$ converges, which is the same as saying that $\gamma\tilde{P}$ is a contraction in some norm.

This follows from the following lemma.

Lemma 6. *Assume that Ξ is a diagonal matrix which has on the diagonal a left eigenvector of P corresponding to eigenvalue one. Then the matrix $\tilde{P} = (\Phi^\top \Xi \Phi)^{-1} \Phi^\top \Xi P \Phi$ is a non-expansion in some norm.*

Proof. Consider first the case when we have $\Xi > 0$. We know that ΠP is a contraction in the norm weighted by Ξ i.e. $\|\Pi P\|_\Xi = \|\Xi^{\frac{1}{2}} \Pi P \Xi^{-\frac{1}{2}}\|_2 \leq 1$ (see for example [32], proposition 6.3.1). Therefore the spectral radius of ΠP is bounded by one. Define the matrix $\Pi^{-} = (\Phi^\top \Xi \Phi)^{-1} \Phi^\top \Xi$, so that we have $\Pi P = \Phi(\Pi^{-} P)$ and $\tilde{P} = (\Pi^{-} P)\Phi$. Using the assumption that Φ has independent columns, it is easy to see that if v is an eigenvector of $\Pi^{-} P \Phi$ then Φv is an eigenvector of $\Phi(\Pi^{-} P)$ with the same eigenvalue. Hence all eigenvalues of $\tilde{P}$ are also eigenvalues of ΠP and $\rho(\tilde{P}) \leq 1$. Now if we have some zero entries on the diagonal of Ξ, our result follows by a continuity argument. Thus we have the result for the general case. $\square$

Note that in the above proof we used the fact that Ξ is a diagonal matrix, where the diagonal entries come from a left eigenvector of P corresponding to eigenvalue one. If Ξ used for the projection were an *arbitrary* distribution, then the matrix $\tilde{P}$ would in general have spectrum beyond the unit circle. For example, consider the following.

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \Phi = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

Now, if we assume a uniform distribution Ξ, we obtain the following matrix, the spectral radius of which is more than one.

$$\tilde{P} = \frac{1}{3} \begin{bmatrix} 2 & 3 \\ 2 & 0 \end{bmatrix}$$

2.2.3 Interpretation in terms of expectations

We now give the interpretation of the matrices $\tilde{P}$ and $\tilde{R}$ in terms of expectations, which are useful when constructing sample-based versions of the algorithm.

$$\tilde{P} = (\Phi^\top \Xi \Phi)^{-1} \Phi^\top \Xi P \Phi = \mathrm{E}\big[\phi_s^\top \phi_s\big]^{-1} \mathrm{E}\big[\phi_s^\top \phi_s'\big]$$

$$\tilde{R} = (\Phi^\top \Xi \Phi)^{-1} \Phi^\top \Xi R = \mathrm{E}\big[\phi_s^\top \phi_s\big]^{-1} \mathrm{E}\big[\phi_s^\top r\big]$$

We also can construct sample-based variants of the matrices $\tilde{P}$ and $\tilde{R}$ (call them $\hat{P}$ and $\hat{R}$ respectively) and still obtain essentially the same algorithm. Let us adopt the following definitions.

$$\hat{\tilde{P}} = \operatorname*{argmin}_{\hat{\tilde{P}}} \|\hat{\Phi}\hat{\tilde{P}} - N\hat{\Phi}\| = (\hat{\Phi}^\top \hat{\Phi})^{-1}(\hat{\Phi}^\top N \hat{\Phi})$$

$$\hat{\tilde{R}} = \operatorname*{argmin}_{\hat{\tilde{R}}} \|\hat{\Phi}\hat{\tilde{R}} - \hat{R}\| = (\hat{\Phi}^\top \hat{\Phi})^{-1}(\hat{\Phi}^\top \hat{R})$$

In the above, we denote by N the matrix which has ones above the main diagonal and zeros elsewhere.

$$N = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & 0 & 1 \\ & & & 0 \end{bmatrix}$$

We note that the required inverse exists by the assumption that $\Phi^\top \Xi \Phi$ is invertible (we also implicitly assume that we have enough samples). Furthermore, we note that the transition model and the reward model are independent, i.e. our solution is also applicable to the setting where we have a single transition matrix P, but instead of just one reward we have many tasks, each of which with a different reward [33]. We note that in this setting, while we still have to learn $\tilde{R}$ for each task separately, it is worthwhile to learn $\tilde{P}$ using training data from *all* tasks.

2.3 Other Ways to obtain LSTD

We begin with the Bellman equation, which defines the true value function: $V(s) = \mathrm{E}[r + \gamma V(s') \,|\, s] = \mathrm{E}[r \,|\, s] + \gamma\, \mathrm{E}[V(s') \,|\, s]$. It can be rewritten in matrix form as $V = (I - \gamma P)^{-1}R$. We exploit a linear architecture: i.e. we seek to approximate the true value function $V(\cdot)$ with the function $\breve{V}(s) = \tilde{V}^\top \phi$, which is linear in $\tilde{V}$. We will briefly discuss two possibilities for how to choose an appropriate $\breve{V}(\cdot)$ within the linear class of functions. The obvious thing would be to proceed along the lines of section 1.4.2 of chapter 1 and define $\bar{V} = \Pi V = \Pi(I - \gamma P)^{-1}R$, where $\Pi = \Phi(\Phi^\top \Xi \Phi)^{-1}\Phi^\top \Xi$ where we assume that the inverse exists. This formula guarantees that the distance from $\bar{V}$ to V is minimal in the elliptic norm weighted by Ξ. The problem with this approach is that it is not known how to efficiently compute a useful estimate of the projected value function from samples.[1] Therefore we need a different approximation. We call it $\breve{V}(\cdot)$. It comes through the equation $\breve{V} = \Pi T \breve{V}$, where we look for the fixpoint of the operator ΠT instead of the Bellman operator T, where $Tx = R + \gamma P x$. Our choice of $\breve{V}$ will be motivated further later (in particular, see equation 2.7).

Now the question, of course, is about the relation between our approximation $\breve{V}$ and the projection of the true value function $\bar{V}$, as we have defined it in the previous section. We now state without proof the relation between the two estimates developed in [34] (see their references for prior work).

$$V - \breve{V} = (I - \gamma \Pi P)^{-1}(V - \bar{V}) \tag{2.3}$$

This can be used to obtain the following bound, which does not require us to estimate the matrices Π or P (see [34] for proof and for sharper bounds): $\|V - \tilde{V}\|_\Xi \leq (1 - \gamma^2)^{-1/2}\|V - \bar{V}\|_\Xi$.

We see from this that one example where the approximation of equation 2.10 is appropriate is when we have substantial discounting – in that case, if the linear framework is good at all, i.e. if the projection $\bar{V} = \Pi V$ is close to the true value function, then so will be our approximation. We emphasise here that our derivation is for the case where there is one recursive class in the MRP. If there are other classes, this bound tells us nothing about them (i.e. using this bound only, we have to accept the value function at the states belonging to them may be arbitrarily off the mark).

[1]One algorithm that can do that in the limit of infinitely many samples is Least-Squares Monte-Carlo. It is, however, prone to high variance in the estimate for small sample sizes.

In the subsequent sections, we will describe various seemingly different approaches to computing $\breve{V}(\cdot)$ from samples, which however all lead to the same formula for the solution we have already seen in section 2.2.2. In order to derive our algorithm, we make two assumptions. First, we assume the following. We call this the *feature independence assumption.*

$$\mathrm{E}\big[\phi^\top \phi\big] \text{ is full rank} \tag{2.4}$$

This implies that the features are linearly independent (i.e. Φ is of full column rank) but the statement is stronger in that it concerns *both* the features and the transition dynamics of the MRP, and means that the parts of the features corresponding to states visited with nonzero probability are independent. We note that this implies that the matrix $\mathrm{E}\big[\phi^\top(\phi - \gamma\phi')\big]$ is also full rank – we discuss why this implication holds in appendix A. We will also use this to claim the invertability of $\hat{\Phi}^\top S \hat{\Phi}$ without further comment (i.e. we assume we have enough samples). Also, we assume that the mean of the reward process exists.

$$\mathrm{E}[r_s] < \infty \tag{2.5}$$

The second assumption is rarely a problem because in typical applications the reward is bounded by some constant.

To summarize the description, we restate the fundamental conditions for LSTD to yield good value estimates: (1) the linear architecture itself needs to match the problem and the set of features needs to be set right, that is V must be close to $\bar{V}$, (2) the approximation $\breve{V}$ needs to be good, for example through discounting and finally (3) the sample based approximation $\hat{\breve{V}}$ to w must also be good (in the following sections we define a consistent estimator for w, i.e. a way to compute $\hat{\breve{V}}$, so that the value function computed from a sample trajectory approaches $\breve{V}$ for the recursive states in the class corresponding to that trajectory as the length of the trajectory goes to infinity). We note as a sideline that if we have the perfect compositionality property of chapter 1, the first point becomes irrelevant.

2.3.1 Derivation by the Galerkin method

That LSTD corresponds to a special case of the Galerkin argument has been implicitly realized for some time, and formally stated in [35], on which this section is based. The general idea of the Galerkin method is to approximate the fixed point of T, $Tx^\star = x^\star$. We have $x^\star = \operatorname{argmin}_x \|Tx^\star - x\|$. We introduce the approximation by considering points from within the column space of Φ, so that our approximate fixpoint satisfies $\tilde{x}^\star \in \text{Range}(\Phi)$, yielding $\tilde{x}^\star = \operatorname{argmin}_{x \in \text{Range}(\Phi)} \|T\tilde{x}^\star - x\|$, which is equivalent to the following, after substituting $\Phi y^\star$ for $\tilde{x}^\star$ and Φy for x and using the semi-norm weighted by Ξ.

$$\Phi y^\star = \operatorname*{argmin}_y \|T\Phi y^\star - \Phi y\|_\Xi \tag{2.6}$$

Now, for our semi-norm with the corresponding projection operator Π, this has an analytic solution: $\Phi y^\star = \Pi(T(\Phi y^\star))$. Now, in our case, $\Pi = \Phi(\Phi^\top \Xi \Phi)^{-1}\Phi^\top \Xi$ where we note that the inverse is well-defined by assumption 2.4 and the evaluation of the operator T at $\Phi\tilde{V}$ becomes $R + \gamma P \Phi \tilde{V}$ with $\tilde{V}$ assuming the role of $y^\star$. Now we solve the following.

$$\Phi\tilde{V} = \Pi(\underbrace{R + \gamma P\Phi\tilde{V}}_{T\Phi\tilde{V}}) \tag{2.7}$$

This can be transformed in the following way.

$$\cancel{\Phi}(I - \gamma(\Phi^\top \Xi \Phi)^{-1}\Phi^\top \Xi P \Phi))\tilde{V} = \cancel{\Phi}(\Phi^\top \Xi \Phi)^{-1}\Phi^\top \Xi R \tag{2.8}$$

In the above, we can cancel out the terms Φ, because by assumption 2.4, Φ has to be of full column rank. We then multiply both sides by $(\Phi^\top \Xi \Phi)$, to obtain $\big((\Phi^\top \Xi \Phi) - \gamma \Phi^\top \Xi P \Phi\big)\tilde{V} = \Phi^\top \Xi R$, which leads to the following.

$$\tilde{V} = \left(\Phi^\top \Xi \Phi - \gamma \Phi^\top \Xi P \Phi\right)^{-1} \Phi^\top \Xi R \tag{2.9}$$

This is the same as the expression we will obtain in the instrumental variable section. We also see that equation 2.8 is the same as the formula obtained from the linear dynamical system approach in equation 2.2 when we plug in the computed values of $\tilde{P}$ and $\tilde{R}$. Thus we have obtained the same estimator.

2.3.2 Derivation by instrumental variables

Again, we begin with the Bellman equation, which defines the true value function: $V(s) = \mathrm{E}[r + \gamma V(s') \,|\, s] = \mathrm{E}[r \,|\, s] + \gamma \,\mathrm{E}[V(s') \,|\, s]$. We will first obtain a statistical model that expresses the properties of the approximation $\breve{V}(\cdot)$. By solving the Bellman equation directly in the linear approximation regime, we obtain the following equation.

$$\phi\tilde{V} = \breve{V}(s) = \mathrm{E}[r \,|\, s] + \gamma \,\mathrm{E}\Big[\breve{V}(s') \,\Big|\, s\Big] - e_s = \mathrm{E}[r \,|\, s] + \gamma \mathrm{E}[\phi' \,|\, s]\tilde{V} - e_s \tag{2.10}$$

We note that in the above, we use the convention that $\tilde{V}$ is a column vector while the features are row vectors. This convention minimizes the number of transposes we have to write. Note that we had to introduce the TD error vector $e = [e_{s_1}, \cdots, e_{s_n}]^\top = T\Phi\tilde{V} - \Phi\tilde{V}$ and the corresponding random variable e_s (i.e. the error is a deterministic function of the current state, which is random), since the sum of the reward vector R and the expected feature vector $\mathrm{E}[\phi' \,|\, s]\tilde{V}$ may not be in the feature space (i.e. the column space of Φ). It can be verified using equation 2.9 that, the error terms satisfy $e\Xi\Phi = 0$, i.e. it is orthogonal to the feature space (indeed it can be seen after a brief manipulation that the condition $e\Xi\Phi = 0$ is *equivalent* to the formula 2.9 – we will do this in section 2.3.4), and that consequently we have the following.

$$\Pi e = 0 \tag{2.11}$$

This is not a derivation from first principles, since we had to use an external argument to verify that $e\Xi\Phi = 0$ (which is equivalent to assuming that the TD error vanishes in expectation). But given the model of equation 2.10 it is nonetheless instructive to look at the mechanics of how the derivation works because this is the first one to have been proposed for LSTD.

We now accept equation 2.10 as a given and give a statistical derivation as provided in the original LSTD paper [17], based on methods described in [36]. Now, because we do not observe the expectations $\mathrm{E}[\gamma V(s') \,|\, s]$ and $\mathrm{E}[r \,|\, s]$

in equation 2.10, but merely samples of ϕ and ϕ' we model the residue wrt. the expected value as noise, yielding the probabilistic model $r_s = \mathrm{E}[r \,|\, s] + \eta_s$, where we use assumption 2.5, and $\phi'_s = \mathrm{E}[\phi' \,|\, s] + \varepsilon_s$. Note that by definition $\mathrm{E}[\eta_s \,|\, s] = 0$. Observe that this implies the following by the law of iterated expectation (LIE).

$$\mathrm{E}[\eta_s \,|\, \phi] = \mathrm{E}[\mathrm{E}[\eta_s \,|\, s] \,|\, \phi] = 0 \tag{2.12}$$

Analogously, we have the following.

$$\mathrm{E}[\varepsilon_s \,|\, \phi] = 0 \tag{2.13}$$

Thus we can rewrite equation 2.10 to obtain the following.

$$\phi\tilde{V} = r_s + \gamma\phi'_s\tilde{V} - \gamma\varepsilon_s\tilde{V} - \eta_s - e_s \quad \text{or} \quad r_s = (\phi - \gamma\phi'_s)\tilde{V} + \underbrace{\gamma\varepsilon_s\tilde{V} + \eta_s}_{\zeta_s} + e_s \tag{2.14}$$

Now, we cannot use traditional least-squares to solve this, since the expression $\zeta_s = \gamma\varepsilon_s\tilde{V} + \eta_s$ may be, in general, correlated[2] with $\phi - \gamma\phi'_s$, so will be the projection error term e and the two correlations will not cancel in general. Therefore the noise term $\zeta_s - e_s$ may be correlated with with $\phi - \gamma\phi'_s$. Also, $\mathrm{E}[e_s \,|\, s]$ is not necessarily zero. But ordinary least squares (OLS) requires that noise be uncorrelated with input variables and that it have mean zero to yield consistent estimates. However, there is still a way to obtain a good estimate. More formally, we first need to establish the following properties. First, we have $\mathrm{E}\big[\phi^\top\eta_s\big] = \mathrm{E}\big[\mathrm{E}\big[\phi^\top\eta_s \,|\, \phi\big]\big] = \mathrm{E}\big[\phi^\top\,\mathrm{E}[\eta_s \,|\, \phi]\big] = 0$, where the first equality follows from LIE and the second from fact 2.12. By the same reasoning, we have $\mathrm{E}\big[\phi^\top\varepsilon_s\big] = 0$ from fact 2.13. With these two properties, we can now multiply both sides of equation 2.14 by $\phi^\top$, which we for this purpose call an *instrumental variable*, and then take expectation, so as to make the noise terms vanish. We also have $\mathrm{E}\big[\phi^\top e_s\big] = 0$ by fact 2.11. This results

[2] Indeed, we have $\mathrm{E}\big[\phi'^\top_s\eta_s\big] = 0$, $\mathrm{E}\big[\phi^\top\varepsilon_s\big] = 0$ and $\mathrm{E}\big[\phi^\top\eta_s\big] = 0$ as shown later in the text; but $\mathrm{E}\big[\phi'^\top_s\varepsilon_s\big] = \mathrm{E}\big[\phi'^\top_s\phi'_s\big] - \mathrm{E}\big[\phi'^\top_s\,\mathrm{E}[\phi'_s \,|\, s]\big] = \Phi^\top\Xi\Phi - \Phi^\top P^\top\Xi P\Phi$, where the last term does not vanish in general.

in the following.

$$\mathrm{E}\big[\phi^\top r_s\big] = \mathrm{E}\big[\phi^\top(\phi - \gamma\phi'_s)\big]\tilde{V} + \underbrace{\gamma\mathrm{E}\big[\phi^\top\varepsilon_s\big]\tilde{V} + \mathrm{E}\big[\phi^\top\eta_s\big] - \mathrm{E}\big[\phi^\top e_s\big]}_{=0} \tag{2.15}$$

Now because we know by assumption 2.4 (see section A of the appendix for a detailed proof) that $\mathrm{E}\big[\phi^\top(\phi - \gamma\phi'_s)\big]$ is invertible, the estimator $\tilde{V}$ is given by the following.

$$\begin{aligned}&\tilde{V} = \mathrm{E}\big[\phi^\top(\phi - \gamma\phi'_s)\big]^{-1}\mathrm{E}\big[\phi^\top r_s\big] = \big(\Phi^\top\Xi(I - \gamma P)\Phi\big)^{-1}\Phi^\top\Xi R \quad \text{or}\\ &\hat{\tilde{V}} = (\hat{\Phi}^\top S\hat{\Phi})^{-1}\hat{\Phi}^\top\hat{r}\end{aligned} \tag{2.16}$$

This finishes the formal derivation. We will now give two different intuitive interpretations to the instrumental variable method. First, consider the sample equivalent of equation 2.14, which we now rewrite in matrix notation $\hat{r} = S\hat{\Phi}\hat{\tilde{V}} + \hat{\zeta} - \hat{e}$, where by $\hat{\zeta}$ we denote the vector containing the noise terms for each individual sample and by $\hat{e}$ the sample values of the random variable e_s. Now, as described above, we cannot solve it by OLS because of the correlation between the noise and $S\hat{\Phi}$. So we 'fix' $S\hat{\Phi}$ by projecting it onto the feature space (i.e. the column space of $\hat{\Phi}$), since we know that noise is uncorrelated with features. We introduce the projection operator $\hat{\Pi} = \hat{\Phi}(\hat{\Phi}^\top\hat{\Phi})^{-1}\hat{\Phi}^\top$, where we note that the inverse exists by assumption given we have enough samples. Now our equation becomes the following.

$$\hat{\Pi}\hat{r} = \hat{\Pi}S\hat{\Phi}\hat{\tilde{V}} + \underbrace{\hat{\Pi}\hat{\zeta}}_{\to 0 \text{ as } N\to\infty} - \hat{\Pi}\hat{e} \quad \text{or} \quad \cancel{\hat{\Phi}(\hat{\Phi}^\top\hat{\Phi})^{-1}}\hat{\Phi}^\top\hat{r} = \cancel{\hat{\Phi}(\hat{\Phi}^\top\hat{\Phi})^{-1}}\hat{\Phi}^\top S\hat{\Phi}\hat{\tilde{V}} \tag{2.17}$$

In the above, we can cancel the terms because $\hat{\Phi}$ has, by assumption, independent columns if we have enough samples. This leads to the same estimator that we derived above. This interpretation is known in econometric literature as two-stage least squares (2SLS), because we solve two linear systems: first we project $S\hat{\Phi}$ on the subspace of features and then we solve the resulting modified equation. In this context we stress that we would get the same solution if we only applied the projection on the right-hand side, e.g. $\hat{r} = \hat{\Pi}S\hat{\Phi}\hat{\tilde{V}}$ – this can be seen by noticing that the choice of $\hat{\hat{V}}$ in this equation is unaffected by any component of $\hat{r}$ orthogonal to the feature space. We also see the direct correspondence between this and the projection step in the derivation through Galerkin method – the equation 2.7 is essentially the limiting version of the sample-based equation 2.17.

2.3.3 The geometry of instrumental variables

There is one more way to interpret the instrumental variable approach. Observe that the equation $\hat{\Pi}\hat{r} = \hat{\Pi}S\hat{\Phi}\hat{\hat{V}}$, can be rewritten as $\hat{\Pi}(S\hat{\Phi}\hat{\hat{V}} - \hat{r}) = 0$. Thus we have that applying the projection amounts to solving $\hat{r} = S\hat{\Phi}\hat{\hat{V}}$ under the constraint that the projection of the residual on the feature space is zero. Therefore LSTD yields the same solution as applying the oblique projection of the rewards on the difference between values of successive states (i.e. $S\hat{\Phi}$), along the subspace orthogonal to the column space of $\hat{\Phi}$ (which is the left null-space of $\hat{\Phi}$). See also figure 2.1.

Recall the formula for the coefficients of the oblique projection on the columns space of X orthogonal to the column space of Y, which is $X(Y^\top X)^{-1}Y^\top$. The corresponding generalized pseudoinverse of X is $(Y^\top X)^{-1}Y^\top$. It is easy to verify that putting $X = (I - \gamma P)\Phi$ and $Y = \Xi\Phi$ into $(Y^\top X)^{-1}Y^\top$ recovers the LSTD solution. Notice that in this case, the projected vector, $X(Y^\top X)^{-1}Y^\top$ corresponds to obtaining the 'smoothed rewards' corresponding to the approximate value function (i.e. $(I - \gamma P)\check{V}S$, or what the rewards would have been if there had been no approximation of the value function). Now there is also a different way of defining the projection, namely we can project not the reward vector but the true value function [25]. In this case, setting $X = \Phi$ and $Y = (I - \gamma P)^\top \Xi\Phi$ again produces the LSTD solution $\tilde{V}$ (note that now, we are projecting the true value function, not the rewards). Notice that in this case the projected vector corresponds to the approximate value function.

Notice that formally speaking, in both the interpretation as a projection of the reward vector and the value function, we also need another condition to call LSTD an oblique projection – in order for the formula $X(Y^\top X)^{-1}Y^\top$ to mean a projection on Range(X) orthogonal to Range(Y), we need the condition that the orthogonal complement of Range(X) and Range(Y) should be complementary subspaces. We will now claim that this is the case in either of the above ways of thinking about LSTD as a projection. To do this, we will prove the following statement. We denote by k the number of columns in Φ (they are known to be linearly independent by assumption 2.4).

Lemma 7. *For any invertible matrices A, B, and Φ of full column rank, we have the following equivalence.*

$$\mathit{Range}(A\Phi)^{\perp} \oplus \mathit{Range}(B\Phi) = \mathbb{R}^n \quad \Leftrightarrow \quad \neg\exists z. \Phi^{\top} A^{\top} B \Phi z = 0$$

Proof. First, we note that the dimension of Range($B\Phi$) is k since $B\Phi$ is full column rank and the dimension of Range($A\Phi)^{\perp}$ is exactly $n-k$ since A is invertible. The argument in the left-to-right direction is as follows: if $\exists z. \Phi^{\top} A^{\top} B \Phi z = 0$, then there would be a vector, $B\Phi z$, which is both in Range($B\Phi$) and Range($A\Phi)^{\perp}$. Therefore these two subspaces cannot sum to the n-dimensional space if they share a common vector. This contradiction finishes the argument. The argument in the right-to-left direction is thus: there is no non-zero vector in both Range($B\Phi$) and Range($A\Phi)^{\perp}$, then because of their dimensions they have to sum to the whole space $\mathbb{R}^n$. □

We now see that the condition $\neg\exists z. \Phi^{\top} A^{\top} B \Phi z = 0$ is fulfilled in the case of LSTD because by assumption 2.4 the matrix $\Phi^{\top} A^{\top} B \Phi$, and hence also $\Phi^{\top} B^{\top} A \Phi$ has to be invertible. In this expression, we can substitute $A = I$ and $B = (I - \gamma P)^{\top} \Xi$ or alternatively $A = I - \gamma P$ and $B = \Xi$ to obtain either of the interpretations of LSTD as projection outlined above. We note that in either case, $B\Phi$ is full column rank by assumption 2.4 together with the fact in appendix A and A is invertible since P is a Markov matrix.

2.3.4 Connection with the iterative TD algorithm

We have seen in section 2.3.2 that the equality $\mathrm{E}\big[\phi^{\top} e_s\big] = -\Phi^{\top} \Xi R + \Phi^{\top} \Xi (I - \gamma P) \Phi \tilde{V} = 0$ is crucial for the development of the algorithm and indeed equivalent to the obtained estimator for $\tilde{V}$ (equation 2.9). We will now show another way of obtaining this equality – actually, it may be taken do be the *definition* of the algorithm, and used as a justification for the formula 2.9 that stands on its own.

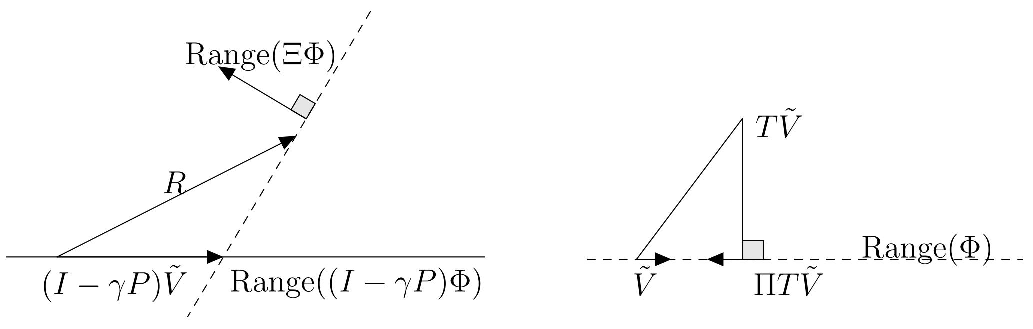


Figure 2.1: LSTD can be interpreted as an oblique projection (left) and as a fixpoint algorithm (right).

We now give the interpretation of this equation is in terms of the iterative TD algorithm [37]. We note that the equality $0 = \mathrm{E}\left[\phi^\top e_s\right]$ corresponds to saying that the LSTD solution corresponds to the fixpoint of iterative TD, i.e. the point where the expected update is zero.

Consider now the definition of the iterative TD algorithm [37]. We assume for the moment that we have an oracle V_o for the value function and are interested in iteratively solving the optimization problem $\min_{\tilde{V}}(V_o(s) - \breve{V}(s))^2$ using the approcimation architecture $\breve{V}(s) = \phi_s\tilde{V}$. The iterative update is given by $\nabla_{\tilde{V}}(V_o(s) - \breve{V}(s))^2 = 2\nabla_{\tilde{V}}\breve{V}(s)\,(V_o(s) - \breve{V}(s))$. We now have the following formula for the iteration.

$$\Delta\tilde{V} \propto \underbrace{\nabla_{\tilde{V}}\breve{V}(s)}_{\phi(s_t)^\top}\underbrace{(\underbrace{(r_{t+1}+\gamma\breve{V}(s_{t+1}))}_{\text{oracle for value}} - \breve{V}(s_t))}_{\text{TD error } e_s}$$

Now we have that the update $\Delta\tilde{V}$ at time t, is $\phi(s_t)^\top e_{s_t}$. Setting the expectation of this update to zero gives the desired formula. We also note that the relation between the TD iteration and the LSTD algorithm resembles the chicken-and-egg problem – one can either, as we did above, consider the iteration a priori knowledge and use that to justify the LSTD fixpoint, or one can start with the fixpoint and treat the iteration as a way of reaching it, motivated by stochastic optimization. LSTD can also be extended to compute the fixpoints of TD(λ) or, more generally other similar algorithms with different traces. For details, see [38] in slightly different notation.

We have seen that the iterative TD algorithm presented above solves the equation $\breve{V} = \Pi T \breve{V}$, which we introduced in section 2.3. However, it should not be confused with another sample-based way of obtaining the fixpoint of the same equation. The other algorithm, which is an adaptation of *fitted value iteration* to policy evaluation, consists of iteratively applying the combined operator ΠT in the sampled regime. More formally, we start with an approximate value function $\mathring{\hat{V}}_k$. We than compute the roll-outs of the operator T from some states, i.e. $T(s) = R_s + \gamma \mathring{\hat{V}}_k(s')$. Assume that a certain number of such roll-outs are stored in a vector $\hat{T}_k$. The value function at the next state $\mathring{\hat{V}}_{k+1}$ can then be computed by projecting the obtained values back on the subspace spanned by Φ i.e. $\mathring{\hat{V}}_{k+1} = \hat{\Pi}_k \hat{T}_k = \Phi \min_{\tilde{V}} \|\Phi \tilde{V} - \hat{T}_k\|$. In practice, this other algorithm is rarely used where we are only interested in policy evaluation rather than solving an MDP with multiple actions.

2.3.5 LSTD as minimization of a quadratic form

This section is based on [39]. It interprets LSTD as the minimization of a quadratic form in the error between the true value function $V(\cdot)$ and the approximated value function $\Phi\tilde{V}$. We begin by reformulating the formula for the estimator obtained above.

$$\begin{aligned}\tilde{V} &= \left(\Phi^\top \Xi (I - \gamma P)\Phi\right)^{-1} \Phi^\top \Xi R = \\ &= \left(\Phi^\top (I-\gamma P)^\top \Xi \Phi (\Phi^\top \Xi \Phi)^{-1} \Phi^\top \Xi (I - \gamma P)\Phi\right)^{-1} \\ &\qquad \Phi^\top (I-\gamma P)^\top \Xi \Phi (\Phi^\top \Xi \Phi)^{-1} \Phi^\top \Xi (I - \gamma P) V\end{aligned}$$

This equality holds because $R = (I - \gamma P)V$ and because the matrices $\Phi^\top (I - \gamma P)^\top \Xi \Phi$ and $\Phi^\top \Xi \Phi$ are invertible by assumption 2.4. Now, we introduce the matrix K, as below.

$$K = (I-\gamma P)^\top \Xi \Phi (\Phi^\top \Xi \Phi)^{-1} \Phi^\top \Xi (I - \gamma P) = (I - \gamma P)^\top \Pi^\top \Xi \Pi (I - \gamma P)$$

We note that $\Xi \Phi (\Phi^\top \Xi \Phi)^{-1} \Phi^\top \Xi = \Xi \Pi = \Pi^\top \Xi = \Pi^\top \Xi \Pi$, where the last equality follows by substituting the definition of Π and cancelling the inverted term. Therefore we have $\tilde{V} = \left(\Phi^\top K \Phi\right)^{-1} \Phi^\top K V$. But this is the solution to the well-known optimization problem: $\tilde{V} = \operatorname{argmin}_{\tilde{V}'} \|V - \Phi\tilde{V}'\|_K = \operatorname{argmin}_{\tilde{V}'}(V - \Phi\tilde{V}')^\top K(V - \Phi\tilde{V}')$. Thus we gain an insight about approximation $\breve{V}(\cdot)$ of equation 2.10 – instead of minimizing the norm $\|\cdot\|_\Xi$, which would yield us $\bar{V}$, we minimize the different norm $\|\cdot\|_K$, thus gaining the ability of efficiently estimating the solution from samples. Note that we can also repeat the above reasoning, without the multiplication by $(\Phi^\top\Xi\Phi)^{-1}$, to obtain the matrix $K' = (I - \gamma P)^\top \Xi\Phi\Phi^\top\Xi(I - \gamma P)$ which also defines a valid minimization – this is the way the equivalence was originally introduced in [27].

2.3.6 LSTD is a subspace algorithm

In section 2.3.3, we have shown that the algorithm can be thought of as an oblique projection along the subspace orthogonal to the feature space. Here, we make explicit the property that LSTD only depends on the features through the subspace they span i.e. any full-rank transformation (i.e. basis change) C of features does not influence the value function. To see this, consider the sample estimate we derived in earlier sections, where we use the transformed features $\hat{\Phi}C$ instead of $\hat{\Phi}$.

$$\begin{aligned}\hat{V}_C &= \hat{\Phi}C\hat{\tilde{V}}_C = \hat{\Phi}C(C^\top\hat{\Phi}^\top S\hat{\Phi}C)^{-1}C^\top\hat{\Phi}^\top\hat{r} = \\ &= \hat{\Phi}CC^{-1}(\hat{\Phi}^\top S\hat{\Phi})^{-1}{C^\top}^{-1}C^\top\hat{\Phi}^\top\hat{r} = \hat{\Phi}(\hat{\Phi}^\top S\hat{\Phi})^{-1}\hat{\Phi}^\top\hat{r} = \hat{V}\end{aligned}$$

As a corollary, we state that LSTD is independent of any scaling of features.

2.4 LSTD vs Bellman Residual Minimization

2.4.1 A Decompositions of the LSTD loss

We now present an interpretation of the minimization defined in equation 2.6, after [22]. We recall that the minimization in equation 2.6 can be rewritten in the following way $\Phi y^\star = \operatorname{argmin}_y \|T\Phi y^\star - \Phi y\|_\Xi = \Pi(T(\Phi y^\star))$. Therefore $\Phi y^\star - \Pi(T(\Phi y^\star)) = 0$, or $\|\Phi y^\star - \Pi(T(\Phi y^\star))\|_\Xi = 0$. Therefore LSTD can be seen to be equivalent to the following optimization problem.

$$y^\star = \operatorname*{argmin}_y \|\Phi y - \Pi(T(\Phi y))\|_\Xi \tag{2.18}$$

We note that this expression has no recursion and that the minimization is guaranteed to reach the optimum value of zero. We can now rewrite the norm as follows $\|\Phi y - \Pi(T(\Phi y))\|_\Xi = \|\Phi y - T(\Phi y)\|_\Xi - \|\Pi(T(\Phi y)) - T(\Phi y)\|_\Xi$, where the equality follows from the Pythagorean theorem and the fact that $\Phi y - \Pi(T(\Phi y))$ and $\Pi(T(\Phi y)) - T(\Phi y)$ are orthogonal vectors, with respect to the Ξ-weighted dot product, which corresponds to Π. We thus obtain the following formula for the LSTD solution.

$$y^\star = \operatorname*{argmin}_y \|\underbrace{\Phi y - T(\Phi y)}_{\text{Bellman residual}}\|_\Xi - \|\Pi(T(\Phi y)) - T(\Phi y)\|_\Xi \tag{2.19}$$

We see that the LSTD algorithm minimizes a quantity which is the Bellman residual minus the reprojection error on the feature space. We discuss in section 2.4.2 the difference between simply minimizing the Bellman residual only and the LSTD algorithm.

Another way to interpret the LSTD loss is to see it as a nested optimization problem [40], which leads to the following two equivalent formulations. First, define the projection in the following way.

$$h^\star(y) = \operatorname*{argmin}_h \|\Phi h - T(\Phi y)\|_\Xi \tag{2.20}$$

Then we plug this for the definition of $\Pi(T(\Phi y))$ in equations 2.18 and 2.19 respectively, giving the following equivalent equations.

$$y^\star = \operatorname*{argmin}_y \|\Phi y - \Phi h^\star(y)\|_\Xi \quad \text{or}$$
$$y^\star = \operatorname*{argmin}_y \left(\|\Phi y - T(\Phi y)\|_\Xi - \|\Phi h^\star(y) - T(\Phi y)\|_\Xi\right) \tag{2.21}$$

2.4.2 Comparison with BRM loss

Instead of constructing the oblique projection as described in the previous sections, we can use a simpler algorithm, known as the Bellman Residual Minimization, which corresponds directly to projecting the rewards on the differences between successive states (see figure 2.2) – i.e. it is similar to LSTD except the projection is orthogonal, not oblique. BRM can be interpreted as the un-nested version of the optimization from the previous section.

$$h^\star = \operatorname*{argmin}_h \|\Phi h - T(\Phi h)\| \tag{2.22}$$

The reason LSTD was originally introduced as an improvement over BRM [17] is that for BRM, we do not have a justification in terms of a statistical model similar to the one we had in section 2.3.2 – the noise terms are correlated, so we cannot use a similar reasoning to claim consistency of BRM. But of course the fact that one line of deriving an algorithm doesn't work for BRM does not mean that the algorithm is wrong – there may be other justifications available. Interestingly, it can be shown that under our assumption 2.4 the two approaches are similar (the argument comes from chapter 4 of [41]). Indeed, we have from the previous section (compare equation 2.18) that LSTD is similar except for the presence of the projection Π. It is sometimes useful to have formulas that make the difference between the two algorithms explicit in different formulations of each algorithm. The algebraic relationships between the two algorithms are summarized in the table below.

LSTD	BRM
$\min_{\tilde{V}} \|\Pi T\Phi\tilde{V} - \Phi\tilde{V}\|_\Xi$	$\min_{\tilde{V}} \|T\Phi\tilde{V} - \Phi\tilde{V}\|_\Xi$
$\min_{\tilde{V}} \|T\Phi\tilde{V} - \Phi\tilde{V}\|_\Xi - \|\Pi T\Phi\tilde{V} - T\Phi\tilde{V}\|_\Xi$	$\min_{\tilde{V}} \|T\Phi\tilde{V} - \Phi\tilde{V}\|_\Xi$
$\tilde{V} = \left(\Phi^\top \Xi L \Phi\right)^{-1} \Phi^\top \Xi R, \quad L = I - \gamma P$	$\tilde{V} = \left(\Phi^\top L^\top \Xi L \Phi\right)^{-1} \Phi^\top L^\top \Xi R$
$\min_{\tilde{V}'} \|V - \Phi\tilde{V}'\|_{(I-\gamma P)^\top \Pi^\top \Xi \Pi (I-\gamma P)}$	$\min_{\tilde{V}'} \|V - \Phi\tilde{V}'\|_{(I-\gamma P)^\top \Xi (I-\gamma P)}$
$\Phi\tilde{V} = \Pi T\Phi\tilde{V}$	$\Phi\tilde{V} = \underbrace{\Phi(\Phi^\top L^\top \Xi \Phi)^{-1}\Phi^\top L^\top \Xi}_{\text{oblique projection, see [25]}} T\Phi\tilde{V}$

There has been renewed interest in the analysis of the difference between the two algorithms. One argument [25] is that in an off-line setting (i.e. in the situation when the weighing coefficients are different from the stationary distribution of the MRP, a scenario we do not consider in this thesis) a performance bound can be shown about BRM that is impossible to derive about LSTD [25]; on the other hand LSTD remains widely used in practice.

There is yet one more feature that means that LSTD is preferable to BRM is some practical cases – while with LSTD, as we have shown above, we only need one sequence of samples of features of states and a sequence of samples of reward to obtain an estimate of the value function; but with BRM we need to have two samples of the features of states.

2.4.3 Sample estimate of the BRM value function

We will now show a way to obtain a sample-based estimate $\hat{\tilde{V}}_B$ of the BRM solution, based on section 3.1 of [42]. We want to minimize the expectation $\mathrm{E}\Big[(\phi\tilde{V}_B - \phi'\tilde{V}_B - r)^2\Big]$. We have the sampled features $\hat{\Phi}^1$ and the sampled rewards $\hat{r}$. We also have a second set of sampled features $\hat{\Phi}^2$. The sampled features are produced using the following process: given the trajectory $s_1, s_2, \ldots$, the features in $\hat{\Phi}^1$ are $\phi(s_1), \phi(s_2), \ldots$ while the features in $\hat{\Phi}^2$ correspond to 'alternative' states $s'_2, s'_3, \ldots$ sampled from $P(\cdot|s_1), P(\cdot|s_2), \ldots$. In other words, the features in $\hat{\Phi}^2$ describe where the MRP might also have gone to given a particular previous state. Of course, such sampling is only possible if we have a model of the transition dynamics of the MRP. Now, we can write a sample-based approximation to the expectation given above as $\hat{E} = \frac{1}{N-1}\sum_{i=1}^{N-1}(\hat{\Phi}^1(i)\tilde{V}_B - \gamma\hat{\Phi}^1(i+1)\tilde{V}_B - \hat{r}(i))(\hat{\Phi}^1(i)\tilde{V}_B - \gamma\hat{\Phi}^2(i)\tilde{V}_B - \hat{r}(i))$, where the notation $\hat{\Phi}^1(i)$ means selecting row i of the matrix $\hat{\Phi}^1(i)$ (i.e. the i-th feature in the trajectory). We can now introduce the notation $\hat{\Psi}^1 = \hat{\Phi}^1(1:N-1) - \gamma\hat{\Phi}^1(2:N)$ and $\hat{\Psi}^2 = \hat{\Phi}^1(1:N-1) - \gamma\hat{\Phi}^2(1:N)$, where the colon notation denotes ranges of rows. With this notation, we have that $\tilde{V}_B^\top(\hat{\Psi}^1)^\top\hat{\Psi}^2\tilde{V}_B = \tilde{V}_B^\top(\hat{\Psi}^2)^\top\hat{\Psi}^1\tilde{V}_B = \sum_{i=1}^{N-1}(\hat{\Phi}^1(i)\tilde{V}_B - \gamma\hat{\Phi}^1(i+1)\tilde{V}_B)(\hat{\Phi}^1(i)\tilde{V}_B - \gamma\hat{\Phi}^2(i)\tilde{V}_B)$. It can now be seen after a few rearrangements that $\hat{E} = \frac{1}{N-1}\left(\tilde{V}_B^\top(\hat{\Psi}^1)^\top\hat{\Psi}^2\tilde{V}_B - \hat{r}^\top(\hat{\Psi}^1 + \hat{\Psi}^2)\tilde{V}_B + \hat{r}^\top\hat{r}\right) = \frac{1}{N-1}\left(\frac{1}{2}\tilde{V}_B^\top((\hat{\Psi}^1)^\top\hat{\Psi}^2 + (\hat{\Psi}^2)^\top\hat{\Psi}^1)\tilde{V}_B - \hat{r}^\top(\hat{\Psi}^1 + \hat{\Psi}^2)\tilde{V}_B + \hat{r}^\top\hat{r}\right)$. Taking the gradient with respect to $\tilde{V}_B$ leaves the us with the system $((\hat{\Psi}^1)^\top\hat{\Psi}^2 + (\hat{\Psi}^2)^\top\hat{\Psi}^1)\hat{\tilde{V}}_B = (\hat{\Psi}^1 + \hat{\Psi}^2)^\top\hat{r}$, where we denoted by $\hat{\tilde{V}}_B$ the sample-based BRM solution.

2.5 Regularization

To overcome the problem of over-fitting, the standard procedure is to add a regularization term to the proposed algorithm. There are many ways of doing that.

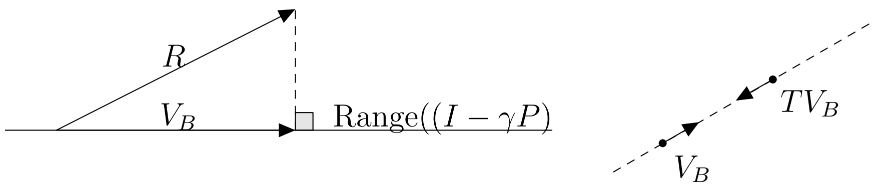


Figure 2.2: BRM as projection of rewards (left) and minimizing the Bellman residual (right). Cmp. fig. 2.1

One way, proposed by [30] is to consider the optimization problem of the fixpoint equation 2.6. We can extend it as follows: $\Phi\tilde{V} = \text{argmin}_{\tilde{V}'}\left(\|R + \gamma P\Phi\tilde{V} - \Phi\tilde{V}'\|_\Xi + \beta\|\tilde{V}'\|\right)$. Here, $\beta \geq 0$ is an external parameter of the algorithm, $\|\cdot\|_\Xi$ is the weighted norm and $\|\cdot\|$ is the usual L_2 norm. This way of regularizing produces the well-known analytic solution $\tilde{V}_R = \left(\Phi^\top\Xi(I-\gamma P)\Phi + \beta I\right)^{-1}\Phi^\top\Xi R$. In the paper [30], a version is also given where the second norm is L_1. In this case, because equation 2.6 is a fix-point equation, it is not possible to simply plug the problem into the standard LASSO algorithm, and a new algorithm is necessary (see [30] for details).

Before we continue, denote the standard L_2-regularized solution of a system of equations $Ax = b$ as $\text{solve}_{L_2}(A, b, \beta) = \text{argmin}_x \|Ax - b\|_\Xi + \beta\|x\|_2 = (A^\top\Xi A + \beta I)^{-1}A^\top\Xi b$. Denote the version with L_1 regularization as $\text{solve}_{L_1}(A, b, \beta) = \text{argmin}_x \|Ax - b\|_\Xi + \beta\|x\|_1$ (this has no explicit analytic form as has to be computed using an algorithm, typically LASSO).

A second way of regularization, introduced in [40] is to add regularization to equation 2.21, giving the following optimization problem.

$$y^\star = \underset{y}{\text{argmin}}\,\|\Phi y - \Phi h^\star(y)\|_\Xi + \|y\|_{1 \text{ or } 2}$$

In the above, the latter norm may be either of L_2 or L_1. A quick calculation shows that this is the same as regularizing the system of equations 2.8. This idea therefore corresponds to the solutions $\text{solve}(\Phi(I - \gamma(\Phi^\top\Xi\Phi)^{-1}\Phi^\top\Xi P\Phi)), \Phi(\Phi^\top\Xi\Phi)^{-1}\Phi^\top\Xi R, \beta)$ for each of the discussed norms.

Another way is adding regularization directly to the equation where we have already solved for $\tilde{V}$, that is, $\tilde{V} = A^{-1}b$, where $A = \left(\Phi^\top\Xi(I-\gamma P)\Phi\right)$ and $b = \Phi^\top\Xi R$. If we regularize with L_2, this corresponds to the solutions $\text{solve}_{L_2}(A, b, \beta)$.

This (together with other versions, that do not map to LSTD), has been done in [41], where the author also derives finite-sample error bounds.

It is also possible to combine some of the above ways together, after the manner of [31], and to use other sparsifiers in place of L_1. In [43], for instance, the Dantzig selector is employed, which leads to a considerable simplification of the optimization problem (the optimization reduces to a linear program).

A yet different approach [44] to regularization is to keep the algorithm itself unchanged and instead do feature selection beforehand. Even if the feature selection algorithm is very simple (greedy based on correlation with residual), simulations [44] suggest that doing feature selection leads to performance essentially the same as the approaches described above. Because greedy feature selection is so simple, this suggests that regularization of LSTD is not yet really a fully solved problem.

A property of all the above regularizers is that we lose the invariance of the algorithm w.r.t. the choice of basis for the feature space, which can be seen as a natural characteristic of LSTD[3]. It is not clear whether the property would be worth preserving in a regularized version – sparsity by its very nature is not invariant to transformations of features, even linear ones and there is a general tendency that a more specialized algorithm will have less generic properties.

2.6 The Episodic version of LSTD

In the other sections of this thesis, we have considered the case where the MRP never terminates and convergence is defined by taking the limit with respect to the length of a trajectory. We are now interested in extending our observations to the case where there is a termination state. The limit will now be the with respect to the number of episodes being accumulated. First, let us note that the formula $\tilde{V} = \mathrm{E}\big[\phi^\top(\phi - \gamma\phi')\big]^{-1}\mathrm{E}\big[\phi^\top r_s\big]$ is still valid in this case. We simply have to give new meaning to the expectation terms.

We will now start by giving a design-based variant for the algorithm. All transitions in a terminating MRP can be described using a rectangular matrix

[3]Indeed section 4 of [31] deals with how to perform standardization of features before plugging them into optimization.

P_t, where the last column is meant to denote termination. We assume in the following that the starting state of the MRP is the first state. We also assume that the matrix P_t is such that the MRP will always eventually terminate. We first need to construct a state distribution Ξ. To do this, we append the row $[1, 0 \ldots 0]$ to the matrix P_t, producing the square matrix P_a, which assumes that the MRP restarts after reaching the termination state. Now, the diagonal entries of the matrix Ξ are the entries of the left eigenvector of P_a which corresponds to eigenvalue one. Now we also construct another square matrix, P, which we obtain by appending the row $[0 \ldots 0, 1]$ to the matrix P_t. This matrix assumes that the agent stays in the termination state forever. The intuition behind this is the following: the matrix P describes the true dynamics of the MRP, but in order to have a meaningful state distribution we need to take into account the fact that we have multiple episodes – hence the definition of the matrix P_a, which models restart. Having defined the above matrices, we may use the standard formula in the following way: $\tilde{V} = \left(\Phi^\top \Xi (I - \gamma P)\Phi\right)^{-1} \Phi^\top \Xi R$. Here, we assume that the last feature vector (i.e. the one corresponding to the state modelling termination) is zero. By definition, the final element of R is also zero.

It can be seen that the sample-based variant is the same as in the case of one long trajectory, except for the additional summation over the episodes. We note we use here the fact that the termination state has the feature of zero (so that we can still use the matrix S – there is no subtraction in the last row, but it doesn't matter since the last state is the terminal state). The formula looks as follows, where the sum goes over episodes.

$$\hat{\tilde{V}} = (\textstyle\sum_e \hat{\Phi}_e^\top S_e \hat{\Phi}_e)^{-1} (\sum_e \hat{\Phi}_e^\top \hat{r}_e)$$

2.7 Summary of Contributions

We have provided a detailed survey of the different ways in which LSTD can be obtained. Our derivation of LSTD using instrumental variables, is, to our knowledge, the first one which is correct. We also made explicit and formal an argument concerning the invertability of the matrix that appears in the LSTD solution (see appendix A). Moreover, we have derived geometric interpretations of the LSTD fixpoint (independently of the work of Scherrer [25], which we only became aware of afterwards). We also provided an exhaustive comparison with the BRM algorithm as well as surveyed the methods that can be used to regularize the LSTD solution. Finally, we formally described the episodic version of LSTD, which was already implicitly known before, but not formalized.

Chapter 3

Options

In this chapter we describe a way of solving MDPs that combines state abstraction and temporal abstraction using the linear framework we developed in chapter 1. Specifically, we combine state aggregation with the options framework and demonstrate that they work well together and indeed it is only after one combines the two that the full benefit of each is realized. We introduce a hierarchical value iteration algorithm where we first coarsely solve sub-goals and then use these approximate solutions to *exactly* solve the MDP. This algorithm solved several problems faster than vanilla value iteration. The approach we take is to modify the well-known value iteration (VI) algorithm of equation 3.3 that we introduced in chapter 1.

In order to solve large problems, table-lookup algorithms are not practical because of the sheer number of states, which VI must loop over. Hence the need for *state abstraction*. For this work, we chose Bertsekas' aggregation framework [10], which is a special case of the compression-decompression scheme of chapter 1 and can be nicely integrated into our framework of the modified Bellman optimality equation. Algorithms based on single-step models of primitive actions are impractical, because long solution paths require many iterations of VI. Hence the need for *temporal abstraction*.[1] We solve this problem via the use of options [46, 47] — we construct option models which can be used interchangeably with the models we have for primitive actions.

We emphasise that even though we use abstraction, the algorithm shown in

[1]Note that there is some evidence [45] that sub-goal-based hierarchical RL is similar to the processes actually taking place in the human brain.

this chapter converges to the *optimal value function* — although we find approximate solutions to the sub-goals, these solutions are then used as inputs to solve the original MDP *exactly*, regardless of the choice of sub-goals.

3.1 State Aggregation in Detail

Before we begin our development, we explicitly describe a specialization of the framework described in chapter one to stochastic matrices. The ideas of this section are entirely due to Bertsekas [10], but we repeat them because the original notation is difficult to apply to our work. Consider an MDP with l actions; for an action a the probability transition matrix is P_a, defined by $P_a(i,j) = \gamma \Pr(i_{t+1} = j | i_t = i, a_t = a)$ and the vector of expected rewards for each state is R_a, where the element corresponding to state i is defined by $R_a(i) = \mathrm{E}\left[r_t | i_t = i, a_t = a\right]$. There are m aggregate states. The two matrices C and D defining the approximation architecture are stochastic. In Bertsekas' notation, the decompression matrix D is the *aggregation matrix* and the compression matrix C the *disaggregation matrix.* The matrix D has dimensions $n \times m$ and the matrix C has dimension $m \times n$. Similar to the reasoning of chapter 1, it is useful to think about these matrices as conversion operators: the matrix D converts a value function defined over the aggregate states into one defined over the original states; conversely, the matrix C converts a value function defined over the original states into one defined over the aggregate states. There are no conditions on these matrices other than non-negativity and the fact that the rows have to sum to one, as they are probability distributions modelling, for D, the degree by which each state is represented by various aggregate states and, for C, the degree to which a certain aggregate state corresponds to various original states. Having defined the matrices, we can define our first approximation step. The Bellman optimality operator T in the original MDP, defined by $(TV)(i) = \max_a (P_a V)(i) + R_a(i)$ and the optimum value function $V^\star$ satisfy the fixpoint equation $V^\star = TV^\star$. Now, the approximation consists in solving the following equation instead (we will see later that this is not solved exactly and further approximation is necessary).

$$\tilde{V}^\star = CT(D\tilde{V}^\star) \tag{3.1}$$

In the above, we use $\tilde{\cdot}$ to denote the aggregate problem. We note that this equation operates on a shorter value function — $\tilde{V}^\star$ has entries corresponding to *aggregate* states. The idea is, of course that the number of aggregate states is tractable, so we can compute $\tilde{V}^\star$. However, we need to reformulate the equation since in its present form it contains the operator T, which still operates on the original states. To do so, we expand the definition of T, to obtain the following state-wise equation, for the aggregate state x.

$$\tilde{V}^\star(x) = \sum_i c_{xi} \left(\max_a e_i^\top P_a D \tilde{V}^\star + e_i^\top R_a \right)$$

This equation leads to the following iterative algorithm, which computes $\tilde{V}^\star$ as $k \to \infty$.

$$\tilde{V}_{(k+1)}(x) = \sum_i c_{xi} \left(\max_a e_i^\top P_a D \tilde{V}_{(k)} + e_i^\top R_a \right)$$

In the above, $e_i^\top P_a$ denotes the row number i of the probability transition matrix corresponding to action a and $e_i^\top R_a$ denotes the corresponding entry of the vector R_a (in terms of the original states). Value functions are assumed to be column vectors. In order to be able to operate exclusively with objects that have dimensionality corresponding to the number of *aggregate* states, we introduce another approximation and namely we do the following.

$$\tilde{V}_{(k+1)}(x) = \max_a \sum_i c_{xi} \left(e_i^\top P_a D \tilde{V}_{(k)} + e_i^\top R_a \right)$$

We note that this approximation is exact if states mapping to a single aggregate state all have the same optimal action. Now, we can reformulate the equation in the following way.

$$\begin{aligned} \tilde{V}_{(k+1)}(x) &= \max_a e_x^\top C P_a D \tilde{V}_{(k)} + e_x^\top C R_a \\ &= \max_a (\tilde{P}_a \tilde{V}_{(k)})(x) + \tilde{R}_a(x) \end{aligned} \tag{3.2}$$

In the above, $e_x^\top C$ denotes the row of C corresponding to aggregate state x and P_a is the probability transition matrix corresponding to action a in the original MDP. Now, we note that solving the above equation is equivalent to solving a modified

MDP with actions corresponding to the original actions, probability transition matrices given by $\tilde{P}_a = CP_aD$ and expected reward vectors given by $\tilde{R}_a = CR_a$. The states of the modified MDP are the aggregate states.

Therefore, under our two explained approximations, solving the original MDP may be replaced by solving a much smaller aggregate MDP, by computing $\tilde{P}_a$ and $\tilde{R}_a$. The solution can then be computed by any known algorithm, although in this thesis we focus only on VI. We emphasize that the VI is *convergent* because the matrices $\tilde{P}_a$ and $\tilde{R}_a$ define a valid MDP. We stress again that this involves two approximations: first, we are solving a modified Bellman equation 3.1 that utilizes state aggregation and second, we move the max operator outside of the sum in equation 3.2.

3.2 Options and Matrix Models

An option [5, 46, 47] is a tuple $\langle \mu, \beta \rangle$, consisting of a policy μ, mapping states to actions, as well as a binary termination condition β, where $\beta(i)$ tells us whether the option terminates in state i. We will now discuss models [46, 5] for options and for primitive actions. A model consists of a transition matrix P and a vector of expected rewards R. For a primitive action a, we defined P_a and R_a in section 3.1. For options they have an analogous meaning. $R(i)$ is the expected total discounted reward given the option was executed from state i, $R(i) = E[\sum_{t=0}^{\tau} \gamma^t r_t | i_0 = i]$ where τ is the (random) duration of the option and i_0 is the starting state. The element $P(i, i')$, is the probability of the option terminating in state i', given we started in state i, considering the discounting: $P_\gamma(i, i') = \sum_{\tau=1}^{\infty} \gamma^\tau \Pr(\tau, i_\tau = i' | i_0 = i)$. Denote by i_0 the starting state of trajectory and by i_τ the final state. As in chapter 1, it is convenient to arrange P and R in a block matrix of size $(n+1) \times (n+1)$, in the following way.

$$\left[\begin{array}{c|c} 1 & 0_\square \\ \hline R & P_\gamma \end{array}\right]$$

3.3 Using Hierarchies to Improve Learning

We give a brief survey of known approaches to hierarchical learning. We stress that our approach is novel for two reasons: we *compose* macro-operators at run-time and we have *no fixed hierarchy.* This has not been done to date, except in the work on options and VI [6], which introduced generalizations of the Bellman equation, versions of which we use. But it did not include state abstraction, slowing the resulting algorithm — it only produced a decrease in the *iteration count* required to solve the MDP, while we provide better *solution time.* Other approaches include using macro-operators to gain speed in planning [48], but for deterministic systems only. Prior work on HEXQ [49] is largely orthogonal to ours – it focuses on hierarchy discovery, while we describe an algorithm *given* the sub-goals. The work on portable options [50] only discusses a flat, fixed (unlike this work) options hierarchy. MAXQ [51] also involves a pre-defined controller hierarchy (the MAXQ graph)[2]. Combining the use of temporal and state abstraction was tried before, but differently from this work. The abstraction-via-statistical-testing approach [53] only works for transfer learning — options are only constructed after the original MDP has been solved. The U-tree approach [54] does not guarantee convergence to $V^\star$ for all MDPs. The modified LISP approach [55] uses a fixed option hierarchy and the policy obtained is only optimal given the hierarchy, i.e. it may not be the optimal policy of the MDP without the hierarchy constraint.

3.4 Table-lookup Value Iteration

We begin by describing the table-lookup (i.e. without aggregation) algorithm for computing the value function of the MDP. We start with plain VI as defined in chapter 1 and then proceed to build up more complicated variants. VI can be described as follows.

$$V_{(k+1)} \leftarrow \text{cmax}\left[\; A_1 V_{(k)} \;\middle|\; \ldots \;\middle|\; A_l V_{(k)} \;\right] \tag{3.3}$$

Here, the operator cmax selects the largest value in each row of the matrix. We rewrite this update to construct a model corresponding to the optimal value func-

[2]One can learn a MAXQ hierarchy [52], but only in a way when it is first learned and then applied.

tion — this is not useful on its own, but will come in handy later.

$$\begin{aligned}\pi &\leftarrow \operatorname{imax}\left[\begin{array}{c|c|c} A_1 M_{(k)} \left[\begin{array}{c} 1 \\ \hline 0_\square \end{array}\right] & \dots & A_l M_{(k)} \left[\begin{array}{c} 1 \\ \hline 0_\square \end{array}\right] \end{array}\right] \\ M_{(k+1)} &\leftarrow A_\pi M_{(k)}\end{aligned} \tag{3.4}$$

Here, the operator imax selects the index of the largest element in each row. We note that the multiplication $M_{(k)}[1, 0, \dots, 0]^\top$ simply extracts the total reward in the model $M_{(k)}$ (the current value function) — hence eq. 3.4 is equivalent to plain VI. However, it serves an an important stepping stone to introducing *sub-goals*, which is what we do next. Assume that we are, for the moment, not interested in maximizing the overall reward. Instead, we want to reach some other arbitrary configuration of states defined by the sub-goal vector G of length $n + 1$. The entry $i + 1$ of G defines the value we associate with reaching state i. We will show later how picking such sub-goals judiciously can improve the speed of the overall algorithm. Our new update, for sub-goal G is the following.

$$\begin{aligned}\pi &\leftarrow \operatorname{imax}\left[\begin{array}{c|c|c} A_1 M_{(k)} G & \dots & A_l M_{(k)} G \end{array}\right] \\ M_{(k+1)} &\leftarrow A_\pi M_{(k)}\end{aligned} \tag{3.5}$$

This iteration converges [6] to a model M_∞, which corresponds to the policy for reaching the sub-goal G. However, this policy executes continually, it does not stop when a state with a high sub-goal value of $G(i + 1)$ is reached. We will now fix that by introducing the possibility of termination — in each state, we first determine if the sub-goal can be considered to be reached and only then do we make the next step. This is a two-stage process, given below. First, we compute the termination condition $\beta(i)$ for each state i, according to the following equation.

$$\beta_{(k)}(i) \leftarrow \operatorname*{argmax}_{\beta_{(k)}(i) \in [0,1]} \beta_{(k)}(i) \left[\begin{array}{c|c} 0 & e_i^\top \end{array}\right] G + (1 - \beta_{(k)}(i)) \left[\begin{array}{c|c} 0 & e_i^\top \end{array}\right] M_{(k)} G \tag{3.6}$$

We note that this optimization is of a linear function, therefore we will either have $\beta_{(k)}(i) = 1$ (terminate in state i), or $\beta_{(k)}(i) = 0$ (do not terminate in state i). Conceptually, this update can be thought of as converting the non-binary sub-goal specification G into a binary termination condition β. Once we have computed $\beta_{(k)}$, we define the diagonal matrix $\beta_{(k)} = \text{diag}(1, \beta_{(k)}(1), \beta_{(k)}(2), \ldots, \beta_{(k)}(n))$ as well as the new matrix B as follows.[3]

$$B(\beta_{(k)}, M_{(k)}) = \beta_{(k)} I + (I - \beta_{(k)}) M_{(k)} \tag{3.7}$$

Here, I is the identity matrix. B summarizes our termination condition — it behaves like model $M_{(k)}$ for the states where we do not terminate and like the identity model for the states where we do. Once we have this, we can define the actual update, which is executed for each state i.

$$\begin{aligned} \pi &\leftarrow \text{imax} \left[\; A_1 B(\beta_{(k)}, M_{(k)}) G \; \middle| \; \ldots \; \middle| \; A_l B(\beta_{(k)}, M_{(k)}) G \; \right] \\ M_{(k+1)} &\leftarrow A_\pi B(\beta_{(k)}, M_{(k)}) \end{aligned} \tag{3.8}$$

By iterating this many times, we can obtain M_∞, which will tend to go from every state to states with high values of the sub-goal G. The elements of G are specified in the same units as the rewards — i.e. this algorithm will go, for the non-terminating states, to a state with a particular value of the sub-goal if the value of being in the sub-goal exceeds the opportunity loss of reward on the way. For the terminating states, the model will still make one step according to the induced policy (see discussion in section 3.2).

There is one more way we can speed up the algorithm — through the introduction of *initiation sets*. In this case, instead of selecting an action from the set of all possible actions, we only select an action from the set of allowed actions for a given state (the initiation set). More formally, let $S_a(i)$ be a boolean vector which has 'true' in the entries where action a is allowed is state i and 'false' otherwise.

[3]The reader will notice that our matrix B can be understood to be the expected model given the termination condition: $B(\beta_{(k)}, M_k) = E_{\beta_{(k)}}[I, M_{(k)}]$. However, in our algorithm it is enough to consider it just a matrix.

Equation 3.8 then becomes the following.

$$\begin{aligned} \pi &\leftarrow \underset{\text{among } S_a(i)}{\text{imax}} \left[\; A_1 B(\beta_{(k)}, M_{(k)}) G \; \middle| \; \dots \; \middle| \; A_l B(\beta_{(k)}, M_{(k)}) G \; \right] \qquad (3.9) \\ M_{(k+1)} &\leftarrow A_\pi B(\beta_{(k)}, M_{(k)}) \end{aligned}$$

In the above, the operator imax is modified to work only on a given subset of each row. The benefit of using initiation sets is that by not considering irrelevant actions, the whole algorithm becomes much faster. We defer the definition of the initiation sets we used to section 3.6.3.

3.5 Combining State Aggregation and Options

We saw in section 3.1 that given the aggregation[4] matrix D and the disaggregation matrix C, we can convert an action with the transition matrix P and expected reward vector R to an aggregate MDP by using $\tilde{P} = CPD$ and $\tilde{R} = CR$. In our matrix model notation, this becomes as follows.

$$\tilde{A} = \left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & C \end{array} \right] A \left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & D \end{array} \right], \text{ where } A = \left[\begin{array}{c|c} 1 & 0_\square \\ \hline R & \gamma P \end{array} \right] \qquad (3.10)$$

This can be viewed as compressing the dynamics, given our aggregation architecture D of size $n \times m$, where m is the number of the aggregate states. We stress that the compressed dynamics define a valid MDP — therefore the algorithms described in the previous section are *convergent*.

The main idea of our algorithm is the following: define a sub-goal, solve it (i.e. obtain a model for reaching it) and then add it to the action set of the original problem and use it as a macro-action, gaining speed. We repeat this for many sub-goals. Solving sub-goals is fast because we do it in the small, aggregate state space. To be precise, we pick a sub-goal $\tilde{G}$ (see section 3.6 for examples) and an approximation architecture D. We then compress our actions with eq. 3.10 and use compressed actions in VI according to 3.8. This gives us a model $\tilde{M}_\infty$ solving the sub-goal in the aggregate state space. We want to use this model

[4]In the work done in this thesis, we used hard aggregation so that each row of D contains a one in one place and zeros elsewhere, and the matrix C is a renormalized version of $D^\top$, so that the rows sum to one.

to help solve the original MDP.

However, we cannot do this directly since our model $\tilde{M}_\infty$ is defined with respect to the *aggregate* state space and has size $(m+1) \times (m+1)$ — we need to find a way to convert it to a model defined over the original state space, of size $(n+1) \times (n+1)$. The new model also has to be *valid*, i.e. correspond to a sequence of actions.[5]

The idea is to make the following transformation: from the aggregate model, we compute the option in the aggregate state space, we then up-scale the option to the original state space, construct a one-step model and then construct the long-term model from it. Concretely, we first compute the *option* corresponding to the model $\tilde{M}_\infty$. The option consists of the policy μ and the termination condition β. We obtain the termination condition by using eq. 3.6 for the aggregate states. The policy μ is obtained greedily for each aggregate state x.

$$\mu = \operatorname{imax} \left[\; A_1 B(\beta, \tilde{M}_\infty)\tilde{G} \; \middle| \; \dots \; \middle| \; A_l B(\beta, \tilde{M}_\infty)\tilde{G} \; \right] \tag{3.11}$$

Now, we can finally build a one-step model in terms of the original state-space. We do this according to the following equation, which we use for each state i.

$$M'(i+1,:) \;=\; (1-\beta(\phi(i))) \; \left[\; 0 \; \middle| \; e_i^\top \; \right] A_{\mu(\phi(i))} \;+\; \beta(\phi(i)) \; e_{i+1}^\top$$

In the above, we denote by e_{i+1} the column $i+1$ of the identity matrix of size $(n+1)\times(n+1)$ and by $\phi(i)$ the aggregate state corresponding[6] to the original state i. In more understandable terms, M' has rows selected by the policy μ wherever the option does not terminate and rows from the identity matrix wherever it does. Now, we do not just need a model that takes us one step towards the sub-goal — we want one that takes us all the way. Therefore, we continually evaluate the option by exponentiating the model matrix, producing M'^∞. Now, this new model still has rows from the identity matrix where the option terminates — therefore

[5]That is why it is not possible to just upscale the model by writing: $\left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0 & D \end{array}\right] \tilde{M}_\infty \left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0 & C \end{array}\right]$.

[6]Note that the equation could be easily generalized to the case where the aggregation is soft — i.e. there are several aggregate states corresponding to i, simply by summing all the possibilities as weighted by the aggregation probabilities.

it does not correspond to a valid combination of primitive actions. To solve this problem, we compute M'', according to the following equation[7] (for each state i).

$$\begin{aligned} M''(i+1,:) &= (1-\beta(\phi(s)))\, M'^{\infty}(i+1,:) + \\ &\qquad \beta(\phi(s))\, A_{\mu(\phi(s))}(i+1,:) \end{aligned} \tag{3.12}$$

M'' contains rows from M'^{∞} where the option does not terminate and rows dictated by the option policy where it does. This guarantees it is a valid combination of primitive actions and can be added to the action set and treated like any other action. We now run value iteration (equation 3.3) using the extended action set — the original actions and the sub-goal models $(M'')^{(q)}$ corresponding to each sub-goal q. This is faster than using the original actions alone, even after factoring in the time used to compute the sub-goal models (see section 3.6).

Observation 4. *Value Iteration with the action set* $\mathcal{A} \cup \{(M'')^{(1)}, \ldots, (M'')^{(g)}\}$ *converges to the optimal value function of the MDP.*

Proof outline. The addition of sub-goal macro-operators to the action set does not change the fixpoint of value iteration because the macro-operators are, by construction, compositions of the original actions. See supplement to existing work [6] for a formal proof of a more general proposition.

This observation tells us that our algorithm will always exactly solve the MDP, computing $V^{\star}$. The worst thing that can happen is that the sub-goal macro-operators will be useless i.e. the resulting value iteration will take as many iterations as without them.

Algorithm 1 provides a pseudocode version of our algorithm combining options and aggregation. In line 2, we generate the action matrices from the problem description. The purpose of the loop in lines 4-13 is to solve each subgoal. In lines 5 and 6, we generate the compression and decompression matrices for the particular subgoal. In line 8 we use them to compress each action. In line 10, we solve the subgoal as outlined in algorithm 2. Here, the convergence criterion is implemented in line 6, where the parameter ε quantifies when we consider two

[7]We note that the infinte exponentiation of a matrix is done as described in appendix B

successive models to be identical. In our experiments, this parameter is taken[8] to equal 0.001. In line 8 of algorithm 1, the variable sol stands for the model for getting to the subgoal in the compressed state space. In lines 11 and 12, we apply the reasoning of this section to produce a model in the original state space, which we store in the variable Macros(s). In line 14, we solve the original MDP by using the initial action set extended by the computed macros.

Algorithm 1 The algorithm combining options and aggregation

```
 1: procedure OPTIONS–AGGREGATION(domain)
 2:     As ← domain.getActions()                              ▷ As = [$A_1, \ldots, A_l$]
 3:     subgoals ← domain.getSubgoals()
 4:     for s = 1, ..., subgoals.length() do
 5:         $D$ ← subgoals(s).aggregation()
 6:         $C$ ← normalize($D^\top$)
 7:         for a = 1, ..., l do                              ▷ Compress each action
 8:             cAs(a) = $\left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & C \end{array}\right]$ As(a) $\left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & D \end{array}\right]$
 9:         end for                                           ▷ cAs = [$\tilde{A}_1, \ldots, \tilde{A}_l$]
10:         sol = ITERATE(cAs,subgoal(s).G)    ▷ Compute model for getting to
                                                 subgoal s
11:         $\langle\mu, \beta\rangle$ ← extractOption(sol)            ▷ Use eqs. 3.11 and 3.6
12:         Macros(s) ← modelFromOption($\mu$,$\beta$,As,$D$)          ▷ Use eq. 3.12
13:     end for
14:     $V^\star$ ← vi([As Macros])               ▷ Solve MDP with enlarged action set
15: end procedure
```

Algorithm 2 The algorithm for solving a single subgoal

```
1: procedure ITERATE (As, G)
2:     newM ← As(1)                           ▷ Use first action as initial model
3:     repeat              ▷ Variable M assumes values $M_{(1)}, M_{(2)}, M_{(3)}, \ldots$
4:         M ← newM
5:         newM(q) ← oneIteration(As, M, G)         ▷ Equations 3.6, 3.7 and 3.8
6:     until ‖(newM − M)G‖ ≤ $\varepsilon$
7:     return M
8: end procedure
```

3.6 Experiments

We applied our approach to three domains: Taxi, Hanoi and 8-puzzle. In each case we compared several variants of VI, including our approach combining state

[8]Although we used this particular value, our experiments are not particularly sensitive to the value of ε as long as the value is sufficiently small.

Figure 3.1: Run-times of our algorithm, plain VI and model VI. All algorithms compute $V^\star$.

Domain	**plain VI**	**model VI**	**options + aggr.**
Taxi (determ.)	6.43 s.	11.64 s.	4.57 s.
Taxi (stoch.)	8.30 s.	47.80 s.	4.83 s.
Hanoi (determ.)	23.45 s.	51.65 s.	11.57 s.
Hanoi (stoch.)	27.31 s.	357.52 s.	21.71 s.
8-puzzle (determ.)	100.19 s.	221.20 s.	85.94 s.

aggregation and options. For vanilla VI we considered algorithms based on both eq. 3.3 (the familiar algorithm, denoted plain VI) and eq. 3.4 (model VI, where complete models are constructed). Figure 3.1 summarises the solution times for each domain; more details are given in the following domain-specific subsections. We, however, stress beforehand that our algorithm produced a speed-up for each of the domains we tried.

3.6.1 The TAXI Problem

TAXI [51] is a prototypical example of a problem which combines spatial navigation with additional variables. Denote the number of states as n (here $n = 7000 + 1$) and the number of aggregate states as m (here $m = 25 + 1$). The one state is the sink state.

In our first experiment, we ran four algorithms computing the same optimal value function, one for each combination of using (or not) state aggregation and options. Consider using neither aggregation nor options — this is model VI, as defined in equation 3.4, which we iterate until convergence. Here, one iteration has a complexity of $\mathrm{O}(n^2 l + n^3)$, in practice it is $\mathrm{O}(nl)$ because of sparsity. It takes 22 iterations to complete.

Now consider the version with sub-goals but no aggregation. Here, we have 5 sub-goals: one for getting to each pick-up location or the fuel pump. We are iterating equation 3.14. An iteration now has complexity $\mathrm{O}(g((l + g)n^2 + n^3))$. Because of sparsity, this becomes $\mathrm{O}(g((l+g)n+n)) = \mathrm{O}(g((l+g)n)$. The algorithm needs 8 iterations less to converge, because sub-goals allow it to make jumps. However, due to the increased cost of each iteration, the time required to converge increased.

Now look at the version with aggregation (see section 3.5) and no options. There are 26 aggregate states. We map each original state to one of 25 states by considering only the taxi position and ignoring other variables. Sink state (state 7001) gets mapped to the aggregate sink state (state 26). We proceed in two stages. This process is demonstrated in algorithm 3. First, in lines 5-7 all actions are compressed (eq. 3.10). Then, in line 8, the problem is solved using model VI (as in equation 3.4) in this smaller state-space. This takes 330 iterations, but is fast because m is small — the complexity is $O(m^2 l + m^3)$. We then obtain the value function of the aggregate system and upscale it, then in line 10 we use the new value function to obtain a greedy model (i.e. each row comes from the action that maximizes that row times the upscaled value function), which we use in line 11 as initialization in a model VI iteration in the original uncompressed problem, which takes 3 iterations less than our original algorithm.

Algorithm 3 The algorithm solving the TAXI domain with aggregation, but without options.

```
1: procedure TAXI–AGGREGATION(domain)
2:     As ← taxi.getActions()                                  ▷ As = $[A_1, \ldots, A_7]$
3:     $D$ ← taxi.aggregation()
4:     $C$ ← normalize($D^\top$)
5:     for a = 1, . . . , l do                                 ▷ Compress each action
6:         cAs(a) = $\left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & C \end{array}\right]$ As(a) $\left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & D \end{array}\right]$
7:     end for                                                 ▷ cAs = $[\tilde{A}_1, \ldots, \tilde{A}_7]$
8:     sol ← model-vi(cAs)                 ▷ Use model iteration of equation 3.4.
9:     Va ← extractValueFunction(sol)
10:    M ← makeGreedyModel( $\left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & D \end{array}\right]$ Va)
11:    return model-vi(As, M)     ▷ Iterate eq. 3.4, initialize iteration with M
12: end procedure
```

Now consider the final version, where the benefits of aggregation and options are combined, which is the same as the process demonstrated in algorithm 1 except we use model value iteration of equation 3.4 in line 14 to make the comparison with other algorithms fair. Again, the algorithm consists of two stages. First, we use compressed actions to compute models for getting to the five sub-goals. This requires 17 iterations; the complexity of each is $O(g(lm^2 + m^3))$, where $g = 5$. This is fast since m is small. We now upscale these models. We see that if we add

Figure 3.2: Run-times of the algorithm in the deterministic and stochastic versions of TAXI .

deter.	**no aggregation**	**aggregation**
no options	22 iter. 11.64 s.	330 + 19 iter. 11.73 s.
options	14 iter. 78.20 s.	17 + 7 iter. 6.55 s.

stoch.	**no aggregation**	**aggregation**
no options	30 iter. 47.80 s.	331 + 28 iter. 26.04 s.
options	18 iter. 256.04 s.	20 + 7 iter. 6.78 s.

the five macro-actions, we do not need the original four actions for moving, as all sensible movement is to one of the five locations. The algorithm now takes only 7 iterations to converge.[9] The run-time[10] is 6.55 s, i.e. a speed-up of 1.8 times over model VI.

Results for all four versions are summarized in figure 3.2. We also constructed a stochastic version of the problem, with a probability of 0.05 of staying in the original state when moving. Results are qualitatively similar and are in figure 3.2. The speed-up from combining options with aggregation was greater at 7.1 times. We stress the main result.[11] In the deterministic case, we replace many O(n) iterations with many O(m^3) iterations followed by few O(n) iterations. For stochastic problems, we replace many O(n^3) iterations with many O(m^3) iterations followed by few O(n^3) iterations.

In our second experiment, as a digression from the main thrust of our reasoning, we tried a different approach: we can use the aggregation framework to compute an *approximate* value function, gaining speed. Our actions are compressed as defined by eq. 3.10, and we simply apply eq. 3.3. This process gives us a value function $\tilde{V}^\star$ defined over the aggregate state space (in the first case

[9] We need an iteration to: (1) go to the fuel pump, (2) fill in fuel, (3) go to passenger, (4) pick up passenger, (5) go to destination, (6) drop off passenger. The 7th iteration comes from the termination condition.

[10] This is slightly different from the result in fig. 3.1 since after the models have been upscaled, we can proceed either with plain VI of equation 3.3 (as is fig. 3.1) or with model VI of equation 3.4, which we do here to make the comparison fair.

[11] The result holds if the number of sub-goals and actions is constant.

we need to extract it from the reward part of the model). We upscale this value function to the original states using the following equation.

$$\bar{V} = \left[\begin{array}{c|c} 1 & 0_{\square} \\ \hline 0_{\square} & D \end{array}\right] \tilde{V}^{\star}$$

Of course, the obtained value function $\bar{V}$ is only *approximately* optimal in the original problem. Consider a D with 501 aggregate states — the aggregation happens by eliminating the fuel variable and leaving others intact. The algorithm used is given by eq. 3.4, applied to compressed actions. It takes 2.94 s / 28 iterations to converge in the deterministic setting and 3.08 s / 30 iterations in the stochastic setting. The learned value function corresponds to a policy which ignores fuel, never visits the pump, but otherwise, if there is enough fuel, transports the passenger as intended. We have shown an important principle — if we have an aspect of a system that we feel our solution can ignore, we can eliminate it and still get an *approximate* solution. The benefit is in the speed-up. — in our case, with respect to solving the original MDP using plain VI, it is 2.2 times in the deterministic setting and 2.7 times in the stochastic setting.

3.6.2 The Towers of Hanoi

For r disks, our state representation in the Towers of Hanoi is an r-tuple, where each element corresponds to a disk and takes values from $\{1, 2, 3\}$, denoting the peg.[12] There are three actions, two for moving the smallest disk and one for moving a disk between the remaining two pegs. It is known that VI for this problem takes 2^r iterations to converge. To speed up the iteration, we introduced the following state abstraction. There are $r - 2$ sub-problems of size 2,...,$r - 1$. First, we solve the problem with 2 disks, i.e. our abstraction only considers the position of the two smallest disks, ignoring the rest. There are three sub-goals, one for placing the two disks on each of the pegs. Then, once we obtained three models for the sub-goals, we use them to solve the sub-problem of size 3, ignoring all disks except the three smallest ones. Again, there are three sub-goals. We

[12]Note that the state representation itself disallows placing a larger disk on top of a smaller one.

proceed until we solve the problem with r disks. For each sub-goal, we need 4 iterations (Three moves and the 4th is required for the convergence criterion). The total number of iterations is $4 \times 3 \times r$, i.e. it is linear in the state space.

The details of this process are shown in algorithm 4, which has been adapted to this particular problem domain. In line 3, we generate the matrices representing the actions in the problem, which we store in the variable As, so that As $= [A_1, \dots, A_l]$. The idea of the loop starting in line 5 is to solve three subgoals related to a subproblem with s smallest disks. In line 7 we generate the decompression matrix that abstracts away all disks other than the s smallest ones, while in line 8 we generate the three subgoals, for placing the s disks on each peg. In lines 9 to 11 we produce macros (i.e. option models) for solving each of the three subgoals. In line 12, we save the computed macros to be used in the next iteration. In line 14, we solve the original problem using the precomputed solution for the problem with disks-1 disks stored in the variable prevMacros. The procedure SOLVESUBGOAL in line 16 works by first compressing all the actions in line 19, then solving the subgoal G in the compressed setting in line 21 and finally constructing an upscaled model of the resulting macro-operator as outlined in section 3.5.

For 8 disks this means the following speed-up: 11.57 s (with sub-goals, i.e. according to algorithm 4) vs. 51.65 s (model VI of equation 3.4) vs. 23.45 s (plain VI of equation 3.3). We note however, that the time complexity of the algorithm with sub-goals is still exponential in r, because whereas the number of iterations is only linear, in each iteration we need to iterate the whole state space, which is exponential.[13] For a stochastic version, the run-times were 357.52 s for model VI, 27.31 s for plain VI and 21.71 s for computing the same optimal value function with options with aggregation.

3.6.3 The 8-puzzle

The 8-puzzle [56, 57] is well-known in the planning community. Our sub-goal is shown in figure 3.3.[14] 'A','B', and 'C' denote groups of tiles. The sub-goal

[13]However, this problem is not particular to our approach — every algorithm that purports to compute the value function *for each state* will have computational complexity at least as high as the number of such states.

[14]Other sub-goals are shown in appendix C. Please also consult the source code, where all sub-goals are implemented.

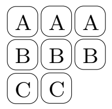
Algorithm 4 The algorithm solving the towers of Hanoi combining options and aggregation.

```
procedure HANOI–OPTIONS–AGGREGATION(disks)
    hanoi ← HanoiDomain(disks)
    As ← hanoi.getActions()                          ▷ As = $[A_1, A_2, A_3]$
    prevMacros ← [ ]                ▷ Stores the solution to previous subproblem
    for s = 2, . . . , disks-1 do      ▷ Solve subproblem with s smallest disks
        hf ← HanoiFeatures(s,disks)
        $D$ ← hf.aggregation()
        ⟨Ga, Gb, Gc⟩ ← hf.generateGs();           ▷ One subgoal for each peg
        newMacros(1) ← SOLVESUBGOAL([As prevMacros], D, Ga)
        newMacros(2) ← SOLVESUBGOAL([As prevMacros], D, Gb)
        newMacros(3) ← SOLVESUBGOAL([As prevMacros], D, Gc)
        prevMacros ← newMacros         ▷ Use macros computed above in
                                         next iteration
    end for
    $V^\star$ ← vi([As prevMacros])         ▷ Solve MDP with enlarged action set
end procedure

procedure SOLVESUBGOAL(As,D,G)
    $C$ ← normalize($D^\top$)
    for a = 1, . . . , length(As) do                     ▷ Compress each action
        $\mathrm{cAs(a)} = \left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & C \end{array}\right] \mathrm{As(a)} \left[\begin{array}{c|c} 1 & 0_\square \\ \hline 0_\square & D \end{array}\right]$
    end for
    sol = ITERATE(cAs,G)                                  ▷ See algorithm 2
    ⟨$\mu$, $\beta$⟩ ← extractOption(sol)             ▷ Use eqs. 3.11 and 3.6
    return modelFromOption($\mu$,$\beta$,As,$D$)               ▷ Use eq. 3.12
end procedure
```

Figure 3.3: The sub-goal used and run-times for the 8-puzzle. All algorithms compute $V^\star$.

	iter.	**time elapsed**
model VI	32	221.20 s.
plain VI	33	100.19 s.
sub-goal	25	109.51 s.
sub-goal w. init. set	25	85.94 s.



consists in arranging the tiles so that each group is in correct place (but tiles *within* each group are allowed to occupy an incorrect place). The matrix D is such that the original configuration of the tiles is mapped onto one where each tile is only marked with the group it belongs to. Using the sub-goal alone did not result in a speed-up, so we used the notion of initiation sets [46]. We trained the sub-goal for 9 iterations (the number 9 was obtained by trial and error), so the obtained model is only able to reach the sub-goal for some starting states (the ones at most 9 steps away from the sub-goal in terms of primitive actions). We upscaled the model, but this time the new model had an initiation set containing only those states from which the sub-goal is reachable. The iteration we then used is plain value iteration, extended to initiation sets. The intuition behind initiation sets is that it only makes sense to use a sub-goal if we are already in a part of the state space close to it. Thus, we obtained a total run-time of 85.94 seconds, which amounts to a speed-up of 1.17 over plain value iteration. The results are in figure 3.3.

3.7 Extensions to the Options framework

In this section, we will develop some possible extensions to the options framework, which are appealing from the theoretical point of view. Ideas in this section are largely based on the paper [6], although we have refined the details after the paper was published (in particular the way in which termination is handled has been changed to be more natural).

3.7.1 Option evaluation in expectation

We have seen in chapter 1 that we can evaluate a policy model A_π by computing A_π^∞. We can extend this operation to terminating option models as follows. Consider an option with the termination condition β and policy π. Consider the matrix $B(\beta, A_\pi)$, defined as in equation 3.7. Now consider the following limiting model.

$$B(\beta, A_\pi)^\infty$$

We observe that the limit exists, since the transition matrix in B is sub-stochastic with the only non-contracting rows being absorbing. We note that unlike in the case of policy evaluation, the transition part of the model may be non-zero.

3.7.2 Simultaneous iteration with many options

Another direction how we can extend our framework is to construct options simultaneously [6], in such a way that at each step, we use all other options to build up the set of options for the next iteration. We will now construct a new algorithm, distinct from the ideas discussed in section 3.5, which can be used to solve

MDPs in a compositional way. We call the algorithm option-option model iteration (OOMI), as it can be viewed as a generalisation of value iteration to option models for multiple sub-goals. Given a base set of option models, and also a set of different sub-goals, the algorithm updates at every iteration the set of current option models, containing one option model for every sub-goal. The maximization is done over the set of actions and the current set of option models. The crucial point is that the algorithm imposes no explicit hierarchy: any option model may be composed with any other option model. When updating the option model for a given sub-goal, *all* current models are considered. In particular, the previous iterate of the option model itself is considered; this allows option models to be repeatedly squared, so that a single model may be efficiently applied as many times as required. As a result, even if OOMI is restricted to primitive actions, and only a single sub-goal, it may still converge in significantly fewer iterations than value iteration. Our algorithm will be a generalization of equation 3.8. We use the current state of every model in every iteration, to compute the next iteration for both itself and other models. Denote our sub-goals by $G^{(1)}, G^{(2)}, \ldots, G^{(g)}$ and the k-th iteration of the models trying to solve these sub-goals by $M^{(1)}_{(k)}, M^{(2)}_{(k)}, \ldots, M^{(g)}_{(k)}$. Define the set $\Omega_{(k)}$ as the set of all models (macro-actions) allowed at iteration k, i.e. $\Omega_{(k)} = \{A_1, A_2, \ldots, A_l, M^{(1)}_{(k)}, M^{(2)}_{(k)}, \ldots, M^{(g)}_{(k)}\}$. This gives rise to the update given below, for each sub-goal q and for each state i. We now compute the termination condition.

$$\beta^{(q)}_{(k)}(i) \leftarrow \underset{\beta_{(k)}(i)\in[0,1]}{\operatorname{argmax}} \; \beta_{(k)}(i) \left[\, 0 \,\middle|\, e_i^\top \,\right] G^{(q)} + (1-\beta_{(k)}(i)) \left[\, 0 \,\middle|\, e_i^\top \,\right] M^{(q)}_{(k)} G^{(q)} \tag{3.13}$$

Then we compute one step of the algorithm according to the equation.

$$\begin{aligned} \pi &\leftarrow \operatorname{imax} \left[\, O_1 B(\beta^{(q)}_{(k)}, M^{(q)}_{(k)}) G^{(q)} \,\middle|\, \ldots \,\middle|\, O_{|\Omega_{(k)}|} B(\beta^{(q)}_{(k)}, M^{(q)}_{(k)}) G^{(q)} \,\right] \qquad (3.14) \\ M^{(q)}_{(k+1)} &\leftarrow O_\pi B(\beta^{(q)}_{(k)}, M^{(q)}_{(k)}) \end{aligned}$$

These updates are used as shown in algorithm 5. The difference from algorithm 2 is that in addition to working with many subgoals simultaneously, in line 6, we include the current iterate of every model in the set of operators considered in the next iteration. Solving several sub-goals simultaneously can improve the algorithm [6]. The immediate availability of the partial solution to every sub-goal leads to faster convergence. In other words, this feature can be used to construct the macro-operator hierarchy at *run time* of the algorithm.[15] This is in contrast to many other approaches, where the hierarchy is fixed before the algorithm is run. Note that the idea of this section can be seen as a generalization of the Bellman optimality equation.

Algorithm 5 The algorithm for simultaneous option-option model iteration

```
1: procedure ITERATE–SIMULTANEOUS (As,Gs)
2:     newMs ← [As(1), ..., As(1)]    ▷ For each of length(Gs) subgoals, use first action as initial model, $M^{(q)}_{(1)} = A_1$
3:     repeat    ▷ Loop generates sequence $[M^{(1)}_{(1)}, \dots, M^{(g)}_{(1)}],[M^{(1)}_{(2)}, \dots, M^{(g)}_{(2)}],\dots$
4:         Ms ← newMs
5:         for q = 1, ..., length(Gs) do
6:             newMs(q) ← oneIteration([As Ms], Ms(q), Gs(q))    ▷ Equations 3.13 and 3.14
7:         end for
8:     until $\sum_q \|(\text{newMs(q)} - \text{Ms(q)})\text{Gs(q)}\| \leq \varepsilon$
9:     return Ms
10: end procedure
```

In table 3.4, we give results from experiments [6], where it is demonstrated that the idea of simultaneous option construction can significantly reduce the number of iteration value iteration takes to converge. The experiments were done using a recursive grid-world domain (nested Nine Rooms).

3.8 Summary of Contributions

We introduced novel Bellman optimality equations that facilitate VI with options. These equations can be combined with state aggregation in a sound way, and therefore can be applied to the solution of medium-sized MDPs. This is the first algorithm combining options and state abstraction which is *guaranteed to converge*. This is significant because other proposed approaches, notably based on linear features, are known to diverge even for small problems. We have also

[15]By this we mean that the option models are built up in run time, possibly using other models. The sub-goals are pre-defined and constant.

Problem size (deterministic)	Plain Value Iteration	Simultaneous Iteration w. Options
2	22	10
3	70	14
4	214	24

Problem size (stochastic)	Plain Value Iteration	Simultaneous Iteration w. Options
2	24	22
3	77	24
4	239	33

Figure 3.4: The number of iterations required with simultaneous option construction as compared to vanilla value iteration in nested nine rooms.

shown experimentally that the benefits of options and state aggregation are only realized when they are applied *together*. Moreover, we have shown that the process of evaluating any of the option models we introduced corresponds to matrix exponentiation. Furthermore, we have described a novel algorithm that learns a set of options on the fly, with each option contributing to all the others and have shown that this idea is very useful in domains which have recursive structure.

Conclusions and Future Work

To summarize, in chapter 1 we have introduced a framework of linear compression that can be used to approach the problem of approximately solving large Markov Decision Processes. We have provided a characterization of what makes the linear features applied to this framework good, as well as a preliminary convergence analysis, which guarantees what we call weak stability. In chapter 2, we analysed the LSTD algorithm and compared it to Bellman Residual Minimization, while also providing geometric interpretations. In chapter 3, we have discussed how a combination of three ideas: linear models, state aggregation and options can produce algorithms that are faster, both in terms of the number of iterations and the actual run-time, than the use of classic value iteration.

We think that the main conclusion from our work is that it is useful to study simple, linear models for Reinforcement Learning. It turns out that even in the linear case, the analysis of algorithms is not trivial and there are valuable insights to be won by studying them. We adopt the philosophical viewpoint that sustainable progress in Reinforcement Learning comes from the careful study of clear-cut models an algorithms rather than from implementing a succession of temporary fixes. Concretely, we believe that this thesis has still fruitful in the following ways: the main result from chapter 1 is that it is possible to define convergent algorithms for solving MDPs in the linear model framework. The main point from chapter 2 is that there are many different ways to derive the LSTD algorithm, which illuminate it in different ways. We have also emphasised the role of geometric intuitions behind the algorithm. Indeed, we are of the opinion that the language of geometry and the associated linear algebra is much more intuitive and easy to understand that the statistical language in which the algorithm was originally formulated. Finally, in chapter 3, we have demonstrated the usefulness of extending value iteration to options, particularly in combination with state aggregation. We have also introduced a useful way of modelling option termination, so that the operation of option evaluation corresponds to simply exponentiating a matrix model.

As for the perspective for the future, we believe that expanding the linear models line of work will eventually lead to very useful applied algorithms. To accomplish that, we think that the following developments are necessary. First, it is necessary to develop reasonably sharp conditions for stability when joint spectral radius of the set of transition matrices is less than one. By sharp we mean that the condition should guarantee convergence to a useful limiting value while still allowing to express the original MDP adequately. Second, if it is impossible to do this while guaranteeing convergence globally, a related direction of further research is to come up with ways of constructing the compression framework C and D, in such a way as to guarantee convergence for the particular MDP that we have. In particular, it would be very fruitful to leverage the work [58, 59] done in the linear systems community on constructing bespoke Lyapunov functions for switched linear dynamical systems. Third, another fruitful avenue of research would be to expand the work done in section 1.7 to maximization over a subset of polices sampled from some distribution. We note that this would immediately make the algorithm practical. Fourth, a very useful line of work would be to infer linear features (what we denote by Φ) automatically just by interacting with the environment. The features, rather than just being required to describe value functions well, would need to satisfy a condition derived from our Riccati equation so that they also fit the dynamics well. Finally, it would be interesting to explore the intersection of our work with Predictive State Representations (PSRs). In particular, PSRs may turn out to be useful way of constructing linear models of the kind described in chapter 1 from samples.

Appendix A

Proof of a fact about equation 2.4 for LSTD

Lemma 8. *Assuming* $\mathrm{E}\big[\phi^\top\phi\big]$ *is invertible, we have that* $\mathrm{E}\big[\phi^\top(\phi-\gamma\phi'_s)\big]$ *is invertible.*

Proof. We rewrite the statement in matrix form: $\det(\Phi^\top\Xi\Phi)\neq 0$ implies $\det(\Phi^\top\Xi(I-\gamma P)\Phi)\neq 0$. We will now develop the second expression. By the well-known eigenvalue argument, $I-\gamma P$ is invertible. Assume for the moment $\Xi>0$ (we will deal with the case when this is not true later). Consider some non-zero vector x. From the assumption $\det(\Phi^\top\Xi\Phi)\neq 0$ we have that $\Phi x\neq 0$. Now, we have that $\Phi^\top\Xi(I-\gamma P)\Phi x=0$ if and only if the vector $y=\Phi x$, which in the column space of Φ satisfies the condition that $\Xi(I-\gamma P)y$ is orthogonal to the column space of Φ. This implies that $y^\top\Xi(I-\gamma P)y=0$. This holds if and only if $y^\top\left(\frac{1}{2}(\Xi(I-\gamma P))+\frac{1}{2}(\Xi(I-\gamma P))^\top\right)y=0$. Now because the matrix defining this quadratic form is symmetric, and thus diagonalizable and with real eigenvalues, we have that this can only be zero if some of the eigenvalues are nonpositive. We will show that this cannot be the case. Rewrite the matrix $\frac{1}{2}(\Xi(I-\gamma P))+\frac{1}{2}(\Xi(I-\gamma P))^\top$ as $\Xi(I-\gamma\frac{1}{2}(P+\Xi^{-1}P^\top\Xi))$. Now because by definition $\Xi=\mathrm{diag}(\xi)$ where $\xi^\top P=\xi^\top$, we have that $\Xi^{-1}P^\top\Xi V1_\square=V1_\square$ (where by $1_\square$ we denote the vector of all ones); moreover, $\Xi^{-1}P^\top\Xi$ has positive entries. So it is a Markov matrix. Thus $\frac{1}{2}(P+\Xi^{-1}P^\top\Xi)$ also is a Markov matrix. Thus, $(I-\gamma\frac{1}{2}(P+\Xi^{-1}P^\top\Xi))$ has eigenvalues in the positive real half-plane. We also know that the eigenvalues of $\Xi(I-\gamma\frac{1}{2}(P+\Xi^{-1}P^\top\Xi))$ are non-negative since it is a symmetric graph Laplacian. But we cannot have zero eigenvalues, because it would imply that $(I - \gamma\frac{1}{2}(P + \Xi^{-1}P^{\top}\Xi))$ also has zero eigenvalues, which we have shown is impossible. This finishes the proof for $\Xi > 0$.

Now consider the case when we do not have this, i.e. some of the diagonal entries of Ξ are zero. Intuitively, the fact we prove is now obvious since transient states do not influence the values of the expectations. More formally, we can, without loss of generality assume that the states for which the probability given by the stationary distribution is zero have highest indexes (i.e. they occur at the back of matrices Ξ, P and Φ). We introduce the following notations for block minors of matrices Ξ, P and the vector y corresponding to the non-transient and transient states.

$$\Xi = \left[\begin{array}{c|c} \Xi^f & 0 \\ \hline 0 & 0 \end{array}\right] \quad P = \left[\begin{array}{c|c} P_f & P_{nt} \\ \hline P_{tn} & P_{tt} \end{array}\right] \quad y = \left[\begin{array}{c} y_f \\ \hline y_t \end{array}\right]$$

Note that in the above, P_{nt} is has to be the zero matrix – it corresponds to transitions from non-transient states to transient states. Therefore we have that $\Xi(I - \gamma P)y = 0$ implies $\Xi^f(I - \gamma P_f)y_f = 0$ and thus, by the reasoning for the case without transient states, y_f has to be the zero vector. Therefore we have the fact that $\Xi(I - \gamma P)\Phi x = 0$ implies that we have the following.

$$\Phi x = y = \left[\begin{array}{c} 0 \\ \hline y_t \end{array}\right]$$

We see that this implies that $\Phi^{\top}\Xi\Phi x = 0$. But we know from our assumption $\det(\Phi^{\top}\Xi\Phi) \neq 0$ that this is only possible for $x = 0$. $\square$

Appendix B

Exponentiating Matrices

We will describe a way of infinitely exponentiating a matrix A, in the sense of $A^\infty = \lim_{n\to\infty} A^n$. The following lemma is due to [60].

Lemma 9. *Assume that the matrix A has eigenvalues one and other eigenvalues strictly within the unit circle. Assume further that if the eigenvalue one occurs t times, there are t distinct eigenvectors corresponding to it. Then we have that $\lim_{n\to\infty} A^n = P\operatorname{diag}(I, 0_\square, \ldots, 0_\square)P^{-1}$, where P is the generalized eigenvector matrix from the Jordan normal form of A, where we assume that the eigenvalues one occur first.*

Proof. Take the Jordan normal form of A, i.e. $A = PJP^{-1}$, where $J = \operatorname{diag}(I, J_1, \ldots, J_m)$, where J_i are Jordan blocks corresponding to eigenvalues less than one and by diag we denote a block diagonal matrix with the specified blocks.

We now claim that $\lim_{n\to\infty} J_i^n = 0_\square$ for $i = 1, \ldots, m$. Consider the block J_i of size $k \times k$. Consider the matrix $D = \operatorname{diag}(\varepsilon, \varepsilon^2, \ldots, \varepsilon^k)$. We have that $\lim_{n\to\infty} J_i^n$ converges if and only if $\lim_{n\to\infty}(D^{-1}J_iD)^n$ converges. The matrix $D^{-1}J_iD$ differs from J_i by having a superdiagonal scaled by ε. Because the moduli of diagonal entries of J_i are less than one, we have $\|D^{-1}J_iD\|_\infty < 1$ for sufficiently small ε.

Because the sup-norm is a submultiplicative matrix norm we have that $\lim_{n\to\infty}(D^{-1}J_iD)^n = 0_\square$. Hence we have that $\lim_{n\to\infty} J_i^n = 0_\square$.

Because we have that $A^n = PJ^nP^{-1}$, we can state the following result.

$$A^\infty = P\operatorname{diag}(I, 0_\square, \ldots, 0_\square)P^{-1}$$

In the above, by $0_\square$ we denote zero blocks of sizes corresponding to $J_1, \ldots, J_m$. $\square$

Appendix C

Software Setup

This note summarizes the steps that have to be taken to reproduce our results on combining value iteration with options with state aggregation. It describes the MATLAB source code used to obtain results discussed in this thesis, as well as gives the output of the software on three separate computers. It also explains how to extend our code to work with other domains.

Hardware and software requirements

We recommend running this software on a computer with at least 2GB of RAM and a processor running at at least 2Ghz. The computer has to be running a recent version of MATLAB (the earliest one we tested was R2012a). The Java heap space allocated to MATLAB must be at least 256MB. In recent versions of MATLAB, this can be set by clicking the 'Preferences' button on the 'Home' tab, then navigating to 'General' and then 'Java Heap memory'. Please note that the default amount that comes when you install MATLAB will typically be less, so you have to change this. We do not support any version of Octave, because it currently does not share MATLAB's support of classes and other object-oriented constructs.

How to run the software

The entry point to this software is the file `run.m`, which is a script that runs all the experiment referred to in this thesis. You can run the software by unpacking the supplied zip archive, changing the current directory to the OptionsAggregation folder and then typing run in the MATLAB console. Please note that executing this file to the end may take some time (a few hours).

Structure of the source code

There are three classes that contain methods that provide a high-level access to the described algorithms, one for each problem domain. These are `TaxiRunner`, `HanoiRunner` and `EightPuzzleRunner`. The methods in these classes then call other functions to finally compute the value function.

There are two pieces of code responsible for the iteration proper. These are the function `iterateAll`, which iterates the algorithm and the class `OptimizePolicyAndTerminationNestedIteration`, which contains the code for the particular kind of two-stage iteration employed in the thesis

The code for generating the subgoal features is contained in classes whose name ends with `Features`. These classes have a method for generating the `Phi` matrix and the `G` vector. Comments in these classes explain the particular kind of approximation employed. For the eight-puzzle problem, there are as many as 15 possible subgoals, which are mapped to the appropriate classes in the function `EightPuzzleRunner.generateSubgoal`. In figure C.1, we graphically show all subgoals referred to in this function.

How to use our algorithm in a new domain

Before we begin, we note that the current version of our software only supports discrete domains where the number of states is manageable (we feel about 200000 states is the maximum that can be currently handled in the deterministic setting, and about 400 states in the completely non-deterministic setting – i.e. where the transition matrices are completely full). Therefore the new domain has to meet these criteria.

The first thing one needs to do is to write a function (or a class) that is capable of generating action models for the new domain. Our system stores actions in cell arrays, so that for instance in the Hanoi problem the `As` vertical cell array has three elements, each a model of size $(n + 1) \times (n + 1)$ where n is the number of states in the MDP. The function `packModel` can be used to convert a vector of expected rewards and a transition matrix into a valid model. That being done, it should be easy to implement value iteration by following steps entirely analogous to the function `TaxiRunner.vi`.

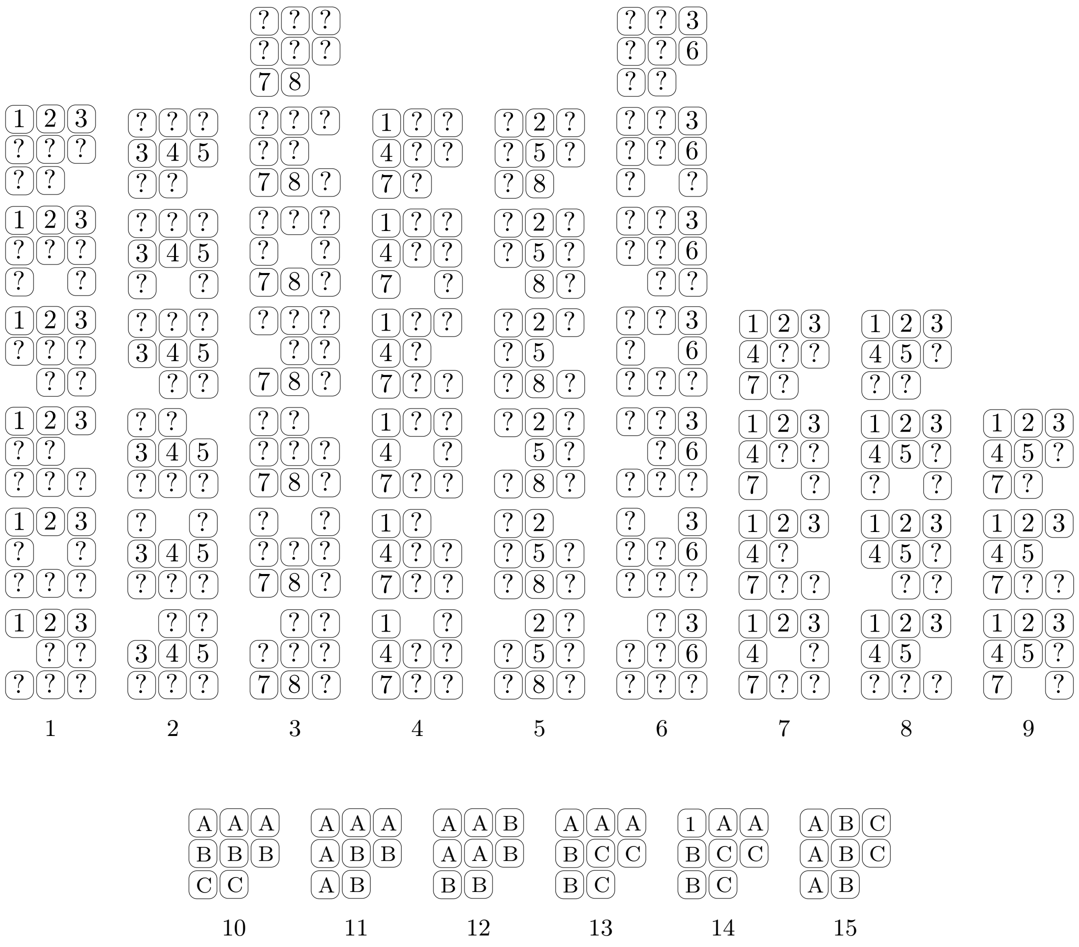


Figure C.1: Subgoals used in the 8-puzzle domain

The next step is to add state abstraction with options. We will describe the case where we have only one subgoal. In this case, we need two things. First, we need a function or a class to generate the aggregation matrix `Phi`, where the rows correspond to original system states and the columns to aggregate states. Then, we also need a function to generate the subgoal `G` in terms of the aggregate states. Once we have these things, it is easy to write code that first solves for the subgoal and then solves the main goal using the subgoal model, based on code in the function `EightPuzzleRunner.optionsAggregation`.

Results of three independent runs

The tables below summarize the results obtained by running the `run.m` script on three separate computers. We begin with the TAXI domain. The 'Iterations' column refers to the iterations solving the main subgoal (we do not give the number of iterations required to solve the subgoals). The time columns refer to the total time necessary to compute the value function, which includes the generation and solution of subgoals. All times are given in seconds. We note the difference between value iteration and plain value iteration – the first one uses matrix models representing the current policy, whereas the plain version only stores the current value function. The number of iterations for the two may be different because of different initialization (model value iteration is initialized with the first action, plain value iteration is initialized with a zero value function).

Problem	Iterations	Time (1)	Time (2)	Time (3)
Deterministic Problem				
Value Iteration (models)	22	11.64	18.36	21.06
Value Iteration (plain)	22	6.43	9.79	9.91
Options	14	78.20	121.06	136.70
Aggregation	19	11.73	18.12	20.11
Options + Aggregation (models)	7	6.55	10.42	11.22
Options + Aggregation (plain)	7	4.57	7.08	7.30
Approx. aggregation (models)	28	4.83	8.29	9.36
Approx. aggregation (plain)	28	2.94	4.92	5.17
Non-deterministic Problem				
Value Iteration (models)	30	47.80	67.40	60.19
Value Iteration (plain)	30	8.30	12.36	12.69
Options	18	256.04	351.73	342.48
Aggregation	28	26.04	38.65	38.02
Options + Aggregation (models)	7	6.78	10.83	11.20
Options + Aggregation (plain)	7	4.83	7.60	7.52
Approx. aggregation (models)	31	5.23	8.82	10.06
Approx. aggregation (plain)	30	3.08	5.22	5.29

The next table gives the results in the Towers of Hanoi domain (with 8 disks).

Problem	Iterations	Time (1)	Time (2)	Time (3)
Deterministic Problem				
Value Iteration (mat. models)	256	51.65	81.49	93.83
Value Iteration (plain)	257	23.45	35.75	39.47
Options with Aggregation	4	11.57	18.97	19.61
Non-deterministic Problem				
Value Iteration (mat. models)	296	357.52	335.32	413.99
Value Iteration (plain)	297	27.31	41.28	45.32
Options with Aggregation	10	21.71	34.64	36.57

Next, we show results for the (deterministic) eight-puzzle domain.

Problem	Iterations	Time (1)	Time (2)	Time (3)
Value Iteration (mat. models)	32	221.20	341.89	395.97
Value Iteration (plain)	33	100.19	155.02	166.87
Options (subgoals 1,2,3)	24	162.52	249.84	269.91
Options (subgoals 1–6)	22	232.39	353.24	386.50
Options (subgoals 1–8)	18	602.71	875.35	917.75
Options (subgoals 7,8)	20	462.91	665.36	696.36
Options (subgoal 7)	26	282.31	411.19	433.51
Options (subgoal 8)	28	299.90	439.42	459.67
Options (subgoal 9)	5	1940.35	1910.67	2422.08
Options (subgoal 10)	25	109.51	169.22	182.50
Options (subgoal 11)	32	130.55	202.01	217.57
Options (subgoal 12)	29	119.33	235.65	198.39
Options (subgoal 13)	32	136.80	210.06	226.88
Options (subgoal 14)	32	153.25	236.72	253.97
Options (subgoal 15)	26	113.65	180.25	189.01
Options (use 1,4 to learn 7, use 1 to learn 8)	20	720.20	1005.39	1033.77
Options (subgoal 10, horizon 9)	25	85.94	131.71	146.08

Bibliography

[1] Sbastien Bubeck and Nicol Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends in Machine Learning*, 5(1):1–122, 2012.

[2] R. Munos. From bandits to Monte-Carlo Tree Search: The optimistic principle applied to optimization and planning. *Foundations and Trends in Machine Learning*, 7(1):1–130, 2014.

[3] Christos Papadimitriou and John N. Tsitsiklis. The complexity of markov decision processes. *Math. Oper. Res.*, 12(3):441–450, August 1987.

[4] Doina Precup. *Temporal Abstraction in Reinforcement Learning.* PhD thesis, University of Massachusetts, Amherst, 2000.

[5] Richard S. Sutton. TD Models: Modeling the World at a Mixture of Time Scales . In *Proceedings of the Twelveth International Conference on Machine Learning*, pages 531–539. Morgan Kaufmann, 1995.

[6] David Silver and Kamil Ciosek. Compositional planning using optimal option models. In *29th International Conference on Machine Learning*, 2012.

[7] M.L. Puterman. *Markov decision processes: discrete stochastic dynamic programming.* Wiley series in probability and statistics. Wiley-Interscience, 2005.

[8] Richard Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, USA, 1957.

[9] G.J. Gordon. *Approximate solutions to Markov decision processes.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1999.

[10] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific Belmont, 2012.

[11] R. Jungers. *The Joint Spectral Radius: Theory and Applications.* Lecture Notes in Control and Information Sciences. Springer, 2009.

[12] Vincent Blondel and Yu Nesterov. Polynomial-time computation of the joint spectral radius for some sets of nonnegative matrices. CORE Discussion Papers 2008034, Universit catholique de Louvain, Center for Operations Research and Econometrics (CORE), 2008.

[13] P. Lancaster and L. Rodman. *Algebraic Riccati Equations.* Oxford science publications. Clarendon Press, 1995.

[14] Ronald Parr, Lihong Li, Gavin Taylor, Christopher Painter-Wakefield, and Michael L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, ICML '08, pages 752–759, New York, NY, USA, 2008. ACM.

[15] Jonathan Sorg and Satinder Singh. Linear options. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1 - Volume 1*, AAMAS '10, pages 31–38, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[16] Martin L Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted markov decision problems. *Management Science*, 24(11):1127–1137, 1978.

[17] Steven J. Bradtke and Andrew G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 1996. 10.1007/BF00114723.

[18] J.A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2):233–246, 2002.

[19] A Nedić and Dimitri P Bertsekas. Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems*, 13(1-2):79–110, 2003.

[20] John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *Automatic Control, IEEE Transactions on*, 42(5):674–690, 1997.

[21] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the twelfth international conference on machine learning*, pages 30–37, 1995.

[22] András Antos, Csaba Szepesvári, and Rémi Munos. Learning near-optimal policies with bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning.*

[23] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, 4:1107–1149, 2003.

[24] R. Munos. Error bounds for approximate policy iteration. volume 20, page 560, 2003.

[25] Bruno Scherrer. Should one compute the Temporal Difference fix point or minimize the Bellman Residual? The unified oblique projection view. In *27th International Conference on Machine Learning - ICML 2010*, Haïfa, Israël, 2010.

[26] Philipp W Keller, Shie Mannor, and Doina Precup. Automatic basis function construction for approximate dynamic programming and reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 449–456. ACM, 2006.

[27] R. Schoknecht. Optimality of reinforcement learning algorithms with linear function approximation. In *Proceedings of the 15th Neural Information Processing Systems conference*, pages 1555–1562, 2002.

[28] David Choi and Benjamin Van Roy. A generalized kalman filter for fixed point approximation and efficient temporal-difference learning. *Discrete Event Dynamic Systems*, 16(2):207–239, 2006.

[29] Manuel Loth, Manuel Davy, and Philippe Preux. Sparse temporal difference learning using lasso. In *Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on*, pages 352–359. IEEE, 2007.

[30] J. Zico Kolter and Andrew Y. Ng. Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 521–528, New York, NY, USA, 2009. ACM.

[31] Matthew Hoffman, Alessandro Lazaric, Mohammad Ghavamzadeh, and Rmi Munos. Regularized least squares temporal difference learning with nested l_2 and l_1; penalization. In Scott Sanner and Marcus Hutter, editors, *Recent Advances in Reinforcement Learning*, volume 7188 of *Lecture Notes in Computer Science*, pages 102–114. Springer Berlin / Heidelberg, 2012.

[32] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, chapter Approximate Dynamic Programming. 2011.

[33] Guy Lever. Private communication.

[34] Huizhen Yu and D.P. Bertsekas. New error bounds for approximations from projected linear equations. In *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, pages 1116 –1123, Sept. 2008.

[35] D. Bertsekas. Temporal difference methods for general projected equations. *Automatic Control, IEEE Transactions on*, (99):1–1, 2011.

[36] J.M. Wooldridge. *Econometric Analysis of Cross Section and Panel Data.* The MIT Press, 2008.

[37] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning.* MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[38] Kamil Ciosek. Generalizing LSTD(λ) to LSTD(λ_t). Internal UCL note., 2012.

[39] Yi Sun, Faustino Gomez, Mark Ring, and Jürgen Schmidhuber. Incremental basis construction from temporal difference error. In Lise Getoor and Tobias Scheffer, editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 481–488, New York, NY, USA, June 2011. ACM.

[40] Matthieu Geist and Bruno Scherrer. l_1-penalized projected Bellman residual. In *European Wrokshop on Reinforcement Learning (EWRL 11)*, Athens, Grèce, 2011.

[41] Bernardo Ávila Pires. Statistical analysis of l1-penalized linear estimation with applications. Master's thesis, Univeristy of Alberta., 2011.

[42] O.A. Maillard, R. Munos, A. Lazaric, and M. Ghavamzadeh. Finite-sample analysis of bellman residual minimization. In *Proceedings of 2nd Asian Conference on Machine Learning (ACML2010)*, Tokyo, Japan, November 2010.

[43] Matthieu Geist, Bruno Scherrer, Alessandro Lazaric, and Mohammad Ghavamzadeh. A Dantzig Selector Approach to Temporal Difference Learning. In *International Conference on Machine Learning (ICML)*, 2012.

[44] C. Painter-Wakefield and R. Parr. Greedy algorithms for sparse reinforcement learning. *Arxiv preprint arXiv:1206.6485*, 2012.

[45] Jose JF Ribas-Fernandes, Alec Solway, Carlos Diuk, Joseph T McGuire, Andrew G Barto, Yael Niv, and Matthew M Botvinick. A neural signature of hierarchical reinforcement learning. *Neuron*, 71(2):370–379, 2011.

[46] Richard Sutton, Doina Precup, and Satinder Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112:181–211, 1999.

[47] Doina Precup, Richard S. Sutton, and Satinder Singh. Theoretical results on reinforcement learning with temporally abstract options. In *Machine Learn-*

ing: ECML-98, volume 1398 of *Lecture Notes in Computer Science*, pages 382–393. Springer Berlin Heidelberg, 1998.

[48] R.E. Korf. *Learning to Solve Problems by Searching for Macro-Operators.* Research Notes in Artificial Intelligence, Vol 5. Pitman, 1985.

[49] Bernhard Hengst. Discovering hierarchy in reinforcement learning with HEXQ . In *International Conference on Machine Learning*, volume 2, pages 243–250, 2002.

[50] George Konidaris and Andrew G Barto. Building Portable Options: Skill Transfer in Reinforcement Learning. . In *International Joint Conferences on Artificial Intelligence*, volume 7, pages 895–900, 2007.

[51] Thomas G Dietterich. The MAXQ Method for Hierarchical Reinforcement Learning. In *International Conference on Machine Learning*, pages 118–126, 1998.

[52] Hongbing Wang, Wenya Li, and Xuan Zhou. Automatic discovery and transfer of maxq hierarchies in a complex system. In *ICTAI*, pages 1157–1162, 2012.

[53] Nicholas K Jong and Peter Stone. State Abstraction Discovery from Irrelevant State Variables. . In *International Joint Conferences on Artificial Intelligence*, pages 752–757, 2005.

[54] Anders Jonsson and Andrew G Barto. Automated state abstraction for options using the U-tree algorithm. *Advances in neural information processing systems*, pages 1054–1060, 2001.

[55] David Andre and Stuart J Russell. State abstraction for programmable reinforcement learning agents. In *AAAI Conference on Artificial Intelligence / Annual Conference on Innovative Applications of Artificial Intelligence*, pages 119–125, 2002.

[56] William E Story. Notes on the "15" puzzle. *American Journal of Mathematics*, 2(4):397–404, 1879.

[57] Alexander Reinefeld. Complete solution of the eight-puzzle and the benefit of node ordering in ida*. In *International Joint Conference on Artificial Intelligence*, pages 248–253, 1993.

[58] Pratik Biswas, Pascal Grieder, Johan Löfberg, and Manfred Morari. A survey on stability analysis of discrete-time piecewise affine systems. In *Proc. 16th IFAC World Congress*, 2005.

[59] Z. Sun and S.S. Ge. *Stability Theory of Switched Dynamical Systems*. Communications and Control Engineering. Springer, 2011.

[60] `http://math.stackexchange.com/questions/378488/limit-of-matrix-powers`. Anonymous post on math forum.