

Adaptive-Resolution Reinforcement Learning with Efficient Exploration in Deterministic Domains*

Andrey Bernstein and Nahum Shimkin

Department of Electrical Engineering

Technion – Israel Institute of Technology

Haifa, Israel

`andreyb@tx.technion.ac.il`

`shimkin@ee.technion.ac.il`

July 22, 2009

Abstract

We propose a model-based learning algorithm, the Adaptive-resolution Reinforcement Learning (ARL) algorithm, that aims to solve the online, continuous state space reinforcement learning problem in a deterministic domain. Our goal is to combine adaptive-resolution approximation scheme with efficient exploration in order to obtain fast (polynomial) learning rates. The proposed algorithm uses an adaptive approximation of the optimal value function using kernel-based averaging, going from coarse to fine kernel-based representation of the state space, which enables to use finer resolution in the “important” areas of the state space, and coarser resolution elsewhere. We consider an on-line learning approach, in which we discover these important areas on-line, using an uncertainty intervals exploration technique. Polynomial learning rates in terms of mistake bound (in a PAC framework) are established for this algorithm, under appropriate continuity assumptions.

1 Introduction

Markov Decision Processes (MDPs) provide a standard framework for handling control and sequential decision problems under uncertainty (Bertsekas, 2007; Puterman, 1994). Solid theory and a variety of algorithms enable the efficient computation of the optimal value function and optimal control policies in MDPs when the state and action spaces are finite. However, an exact solution becomes intractable when the number of states is large or infinite. In this case, some approximation schemes are required. See Boutilier et al. (1999); Bertsekas (2007); Powell (2007), for a general discussion and overview.

A well-established approximation approach is that of value function approximation using a finite set of basis functions, or *kernels*. A particular case is *state aggregation*, in which the state space is represented by a finite (and relatively small) collection of

*Submitted to the Machine Learning Journal.

cells. Each cell is said to *aggregate* the states that fall in this cell. Once the aggregation is performed, the new problem is a planning problem in a reduced state space, which can be solved by regular techniques. The main question that arises here is how to perform the aggregation, so that, on the one hand, obtain a “good” approximation of an optimal policy, and on the other hand reduce the problem complexity. This question was addressed by many works, such as Whitt (1978); Chow and Tsitsiklis (1991), which provide formal answers under some continuity assumptions on the model parameters.

An extra difficulty is added when dealing with *learning problems*, namely situations where the MDP model is initially unknown. Reinforcement Learning (RL) encompasses a wide range of techniques for solving this problem by interacting with the environment. An important part of an RL algorithm is the exploration scheme. The role of exploration is to gain new information by appropriate action selection which directs the agent towards unknown states of the MDP.

Recently, efficient learning algorithms were presented and proved to learn nearly optimal behavior (with high probability) within a time or error bound that is polynomial in the problem size. These include the E^3 (Kearns and Singh, 2002), R-MAX (Brafman and Tennenholtz, 2002), MBIE (Strehl and Littman, 2005), UCRL (Auer and Ortner, 2006), GCB (Chapman, 2007) and OLP (Tewari and Bartlett, 2007) algorithms. These algorithms use efficient exploration techniques, which are often based on the so-called “optimism in face of uncertainty” principle. However, these algorithms are infeasible in cases where the state and/or action spaces are very large or infinite, since their time and space complexity is typically polynomial in the size of the space.

On the other hand, most of the existing algorithms for solving “large” RL problems that rely on function approximation do not provide the analysis of the learning rates, and can be roughly divided into two groups. The first group contains algorithms that are heuristic in nature. These include works on adaptive aggregation such as Moore and Atkeson (1995); Munos and Moore (2002); Bonarini et al. (2007), and general kernel-based algorithms as in Engel et al. (2003); Jong and Stone (2006); Reisinger et al. (2008). It should be noted that in the recent paper by Jong and Stone (2006), an exploration technique is combined with kernel-based function approximation; however no theoretical guarantees were provided for the proposed scheme. The algorithms in the second group avoid the treatment of the exploration problem by explicitly assuming that a (long) sample path of a fixed policy is given a-priori, and the learning is thus reduces to a supervised learning from a “training set” (Ormoneit and Sen, 2002; Loth et al., 2006; Antos et al., 2008). In this case, it is usually shown that as the size of the sample path goes to infinity and provided that this path is “representative” enough, the approximate value function converges to the optimal one. In addition, in the recent paper by Antos et al. (2008), *finite-sample* theoretical guarantees were provided as well.

An important class of control and learning problems deals with *deterministic* models over a continuous (infinite) state space. The examples of such problems and applications of RL algorithms to them are the “Car On Hill” problem (see for instance Sutton (1996); Moore and Atkeson (1995)), the inverted pendulum and cart-pole problems (as in Doya (2000); Sutton (1996)), and the “Acrobot” problem (Sutton, 1996). The applied RL algorithms were heuristic extensions of the classical finite state space algorithms using various kinds of approximation schemes.

In this paper we focus on the online reinforcement learning problem in MDPs with large or infinite state space, finite action space, discounted return criterion, and with *de-*

deterministic dynamics and rewards. Our goal is to combine adaptive-resolution approximation scheme with efficient exploration in order to obtain fast (polynomial) learning rates. For concreteness we will focus on the continuous state case; however our schemes and results also apply to the discrete case, where the number of states is very large or countably infinite. The proposed algorithms use an adaptive approximation of the optimal value function using *kernel-based averaging*, going from coarse to fine kernel-based representation of the state space, which enables to use finer resolution in the “important” areas of the state space, and coarser resolution elsewhere. We consider an on-line learning approach, in which we discover these important areas on-line, using an *uncertainty intervals* exploration technique¹. Certain continuity assumptions on the basic model parameters will be imposed. Such assumptions are essential for generalization in continuous state space.

There is a wide use of the term “kernel” in the literature. Our usage of kernels mostly resembles that of Ormoneit and Sen (2002) in the sense that the kernels are used as local basis functions; however, for computational purposes, we focus on *finite support* kernels rather than a Gaussian-type ones. From this perspective, our kernel-based approach can be compared with *spline*-based approximations of signals, which are recently widely used in signal and image processing field (Unser, 1999). We also note that a similar use of kernel functions in the case of a *finite* state space was proposed by Singh et al. (1995), in the context of soft state aggregation.

The principle that governs our scheme is simply *to split frequently visited kernels*. The idea behind this principle is as follows. As time progresses, we will visit states in the support of kernels that are “close” to the optimal trajectory; and on the optimal trajectory, we need high resolution.

Our focus in this paper is on the combination of the adaptive resolution scheme with efficient exploration that leads to fast (polynomial) learning rates. As a metric for the learning rates in our algorithms we will use the *total mistake count*. This metric counts the total number of time-steps in which the algorithm’s computed policy is strictly suboptimal for the current state. This metric has been used in a number of recent works on on-line learning in discounted MDP problems² (Kakade, 2003; Strehl and Littman, 2005; Strehl et al., 2006b). In our case we will establish two types of mistake bounds, which we call the *prior bound* and the *posterior bound*. The first type ensures that our algorithm is not worse than a non-adaptive algorithm, which uses a single set of *uniformly dense* kernels. In this case, our mistake bound is polynomial in the number of kernels in the latter set. The second type ensures that the total mistake count is polynomial in the number of *actually used kernels* (throughout the algorithm operation). We note that, to our best knowledge, these are the first online performance bound results for kernel-based RL algorithm in continuous state space, for both non-adaptive and adaptive case.

The algorithm proposed in this paper is related to the Adaptive Aggregation Algorithm (AAA) introduced in Bernstein and Shimkin (2008) (see also Bernstein (2007)). In the AAA, constant kernels are used, leading to adaptive state aggregation. Therefore, the value function approximation scheme proposed in this paper is a generalization

¹Our uncertainty intervals are the analogue of the *confidence intervals* used in the stochastic case (Strehl and Littman, 2005). However, the origin of uncertainty in our model is the (deterministic) aggregation error, rather than stochastic sampling error.

²These works refer to this metric as the *sample complexity of exploration*.

of that of the AAA. However, the proposed algorithm itself is different since we have eliminated the need of the so-called “virtually visited cells” by using an improved dynamic programming operator. Hence, the obtained algorithm is simpler than AAA (both conceptually and computationally).

The paper is structured as follows. In Section 2 we present the model and notation. In Section 3 we introduce some further definitions and assumptions. In Section 4 we propose our general algorithm, while Section 5 discusses some computational aspects of the proposed algorithm. In Section 6 polynomial bounds on the total mistake count of the algorithm are presented, while Section 7 proves these bounds. Section 8 introduces an *incremental* variant of our algorithm, which performs only few back-ups of Value Iteration each time-step, instead of exact calculation. Its analysis is presented in Section 9. Finally, conclusions and future work are presented in Section 10.

2 Model and Performance Metric

We denote a deterministic MDP by the 4-tuple $M = (\mathbb{X}, \mathbb{A}, f, r)$, where \mathbb{X} is a state space, \mathbb{A} is an action space, $f(x, a)$ is the transition function which specifies the next state $x' \in \mathbb{X}$ given the previous state $x \in \mathbb{X}$ and action $a \in \mathbb{A}$, and $r(x, a)$ is the immediate reward function which specifies the reward of performing action $a \in \mathbb{A}$ in state $x \in \mathbb{X}$. We assume that $r(x, a) \in [r_{\min}, r_{\max}]$, a finite and known interval.

Let $d : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{R}$ be a fixed metric on \mathbb{X} . We assume the following regarding the state and action spaces.

Assumption 1.

1. The action space \mathbb{A} is a finite set.
2. The state space \mathbb{X} is a bounded subset of \mathbb{R}^n . Thus, there exists a constant $\Delta_{\max} < \infty$ such that for all $x, x' \in \mathbb{X}$, $d(x, x') \leq \Delta_{\max}$.

The MDP M is used to model a dynamic environment, or a dynamic system, with which a learning agent interacts. The interaction proceeds as follows: At time t the agent observes the state $x_t \in \mathbb{X}$, chooses an action $a_t \in \mathbb{A}$, receives a reward $r_t = r(x_t, a_t)$, and the process moves to state $x_{t+1} = f(x_t, a_t)$.

Let $h_t \triangleq \{x_0, a_0, x_1, a_1, \dots, x_{t-1}, a_{t-1}, x_t\}$ denote the *history* of observed states and actions, that is available to the agent at time t to make its choice of a_t . Also, let $\mathbb{H}_t \triangleq (\mathbb{X} \times \mathbb{A})^t \times \mathbb{X}$ denote the space of all possible histories up to time t . Then, at each time t , the agent makes its decision according to some *decision rule* $\pi_t : \mathbb{H}_t \rightarrow \mathbb{A}$, so that $a_t = \pi_t(h_t)$, $t \geq 0$. The collection $\pi = \{\pi_t\}_{t=0}^{\infty}$ is the *control policy*. A policy is *stationary* if the decision rule does not change over time, and depends only on the last state observed. We shall slightly abuse notation and identify the stationary policy π with the map $\pi : \mathbb{X} \rightarrow \mathbb{A}$, so that at each time t , $a_t = \pi(x_t)$.

In this paper we focus on the *discounted return criterion*. For a given initial state $x_0 = x$, we denote the infinite horizon discounted return of state x , for a given policy π , in MDP M , by

$$J_M^\pi(x) \triangleq \sum_{t=0}^{\infty} \gamma^t r(x_t, \pi_t(h_t)),$$

where $0 < \gamma < 1$ is the *discount factor*. The optimal return is denoted by $V_M(x) \triangleq \sup_{\pi} J_M^{\pi}(x)$, which is also called *the optimal value function*. We often drop M from the notation above if it does not cause confusion. A policy π is *optimal* if $J^{\pi}(x) = V(x)$ holds for all $x \in \mathbb{X}$. For any $\epsilon > 0$, a policy π is ϵ -*optimal* if $J^{\pi}(x) \geq V(x) - \epsilon$ holds for all $x \in \mathbb{X}$.

In what follows, we let $V_{max} \triangleq \frac{r_{max}}{1-\gamma}$. Note that V_{max} is the maximal possible discounted return of any policy. Also, let

$$V_b \triangleq \frac{1}{1-\gamma}(r_{max} - r_{min}) \quad (1)$$

denote the maximal difference between the returns of any two policies.

It is well known (Puterman, 1994) that the optimal value function satisfies Bellman's equation

$$V(x) = \max_{a \in \mathbb{A}} \{r(x, a) + \gamma V(f(x, a))\}, \quad x \in \mathbb{X}, \quad (2)$$

and any stationary deterministic policy π^* which satisfies

$$\pi^*(x) \in \operatorname{argmax}_{a \in \mathbb{A}} \{r(x, a) + \gamma V(f(x, a))\}, \quad x \in \mathbb{X},$$

is an optimal policy. Let $Q(x, a) \triangleq r(x, a) + \gamma V(f(x, a))$ denote the action-value function, or Q -function, which provides the return of choosing an action a in state x , and then following an optimal policy.

To assess the learning rate of our algorithms, two performance metrics will be considered. The first and simpler one is the total (action) mistake count, which is given by

$$\text{AMC}(\epsilon) \triangleq \sum_{t=0}^{\infty} \mathbb{I}\{Q(x_t, a_t) < V(x_t) - \epsilon\}, \quad \epsilon \geq 0. \quad (3)$$

This metric counts the number of sub-optimal actions, that is, the number of times that an algorithm executed an action whose action-value is ϵ -inferior to the optimal value. Observe that for $\epsilon = 0$, $\text{AMC}(0)$ is simply the total number of non-optimal actions taken in the course of learning. For deterministic domains with finite state-space, we then have the following near-optimality criterion.

Definition 1 (Action Mistake Bound). *Assume that $|\mathbb{X}|$ is finite. A learning algorithm is PAC (Probably Approximately Correct)³ if there exists a polynomial*

$$B = B\left(|\mathbb{X}|, |\mathbb{A}|, \frac{1}{1-\gamma}, \frac{1}{\epsilon}\right)$$

such that for all $\epsilon > 0$, $\text{AMC}(\epsilon) < B$.

In the above definition, the bound B depends on the number of states $|\mathbb{X}|$. In case $|\mathbb{X}|$ is infinite, some other measures of \mathbb{X} must be considered. As already mentioned, in our case we will replace $|\mathbb{X}|$ by the cardinality of the set of sufficiently dense kernels over the state space.

³Note, that while the ‘‘probably’’ aspect is absent in our deterministic case, we will find it convenient to keep the PAC terminology.

A second metric of interest is the *policy-mistake* count, which counts the number of time steps t in which the algorithm executes a non ϵ -optimal policy. Specifically, let $\pi_t : \mathbb{H}_t \rightarrow \mathbb{A}$ be the decision rule that the algorithm uses at time t to choose its action. Then, given h_t , $\mathcal{A}_t = \{\pi_k\}_{k=t}^\infty$ is a (non-stationary) policy that the algorithm implements at time t , and $\sum_{k=t}^\infty \gamma^{k-t} r_k \triangleq J^{\mathcal{A}_t}(x_t)$ can be interpreted as the return of this policy from time t onward, where $r_k = r(x_k, \pi_k(x_k))$ and $x_{k+1} = f(x_k, \pi_k(x_k))$. Now, the policy-mistake count is defined as

$$\text{PMC}(\epsilon) \triangleq \sum_{t=0}^\infty \mathbb{I} \left\{ J^{\mathcal{A}_t}(x_t) < V(x_t) - \epsilon \right\}. \quad (4)$$

It is easily verified (see Corollary 1 in Bernstein (2007)) that policy-mistake count is a stronger criterion than action-mistake count, in the sense that $\text{AMC}(\epsilon) \leq \text{PMC}(\epsilon)$. Hence we will state all bounds below in terms of the PMC, with the understanding that they apply to the AMC as well.

3 Preliminaries

3.1 Kernel Functions

Our algorithm uses an approximation of the optimal value function $V(x)$ using kernel-based averaging. As was already mentioned, we use the term “kernel” to denote any *local* basis function $\phi : \mathbb{X} \rightarrow \mathbb{R}^+$. We shall focus in this work on *finite support* kernels. Recall that the support of a function $\phi : \mathbb{X} \rightarrow \mathbb{R}^+$ is defined as $\text{supp}(\phi) \triangleq \{x \in \mathbb{X} : \phi(x) \neq 0\}$. Let

$$\Delta(\phi) \triangleq \sup_{x, x' \in \text{supp}(\phi)} d(x, x') \quad (5)$$

denote the *diameter of the support* of ϕ under metric d .

Now, our basic definition is the following.

Definition 2 (Feasible Kernel Set). *A set $\Phi = \{\phi_i\}_{i=1}^m$ of functions $\phi_i : \mathbb{X} \rightarrow \mathbb{R}^+$ is called a feasible kernel set for \mathbb{X} if the following holds:*

1. *Finite support:* $\Delta(\phi_i) < \infty$ for all $i = 1, \dots, m$, and
2. *Uniform coverage:* $\sum_{i=1}^m \phi_i(x) = 1$, for all $x \in \mathbb{X}$,

This definition encompasses several standard types of kernels, such as strict state aggregation, soft state aggregation, splines, etc. See Section 3.2 for further details.

3.2 Splitting Schemes

As we are interested in *adaptive* kernel-based approximation of the optimal value function, we specify the way we perform refinement of a given kernel.

Definition 3 (Uniform Splitting Scheme). *A splitting (or refinement) scheme tells us how to split any given kernel function ϕ into $K \geq 2$ new kernel functions ϕ_1, \dots, ϕ_K .*

We say that the splitting scheme is uniform if there exists $0 < \lambda < 1$ such that for any given kernel function ϕ it holds that

$$\Delta(\phi_j) \leq \lambda \Delta(\phi), \quad j = 1, \dots, K,$$

and

$$\sum_{j=1}^K \phi_j(x) = \phi(x).$$

Now, given a fixed uniform splitting scheme and an initial kernel set Φ_0 for \mathbb{X} , we define the *achievable class of kernel sets* as the set of all kernel sets that can be obtained by repeatedly using the given splitting scheme starting from Φ_0 . Observe that all such kernel sets are feasible by the virtue of Definition 3. We note that both the initial kernel set Φ_0 and the splitting scheme will be fixed in the course of the algorithm.

Several standard types of kernels fit the above definitions. We list below two of such types.

Splines. In spline-based approximations of signals, a one-dimensional kernel (or *spline*) is a scaling and a dilation of some basic finite support function $\varphi^{(k)} : \mathbb{R}^+ \rightarrow \mathbb{R}^+$, that is

$$\phi(x) = A\varphi^{(k)}\left(\frac{x}{b}\right) \text{ for some } b > 0 \text{ and } A > 0.$$

Here, $\varphi^{(k)}$ is called a *basic spline* (or, B-spline) of degree k and is formed by the $(k+1)$ -fold convolution of a rectangular pulse $\varphi^{(0)}(x) \triangleq \mathbb{I}\{|x| \leq 0.5\}$. In particular, $\varphi^{(0)}$ leads to a piecewise-constant approximation (i.e. aggregation), $\varphi^{(1)}$ – to a piecewise-linear approximation, and so on.

One of the advantages of splines is that there exist computationally efficient splitting schemes for them, which fit Definition 3. For example, consider the B-spline of degree 1 shown in Figure 1 (a). A standard B-spline “pyramid” (in the terminology of Unser (1999)) is shown in 1 (b). As it can be observed, the original spline of support Δ was split to *three* splines of support $\Delta/2$, in order to satisfy the condition that the sum of the new splines equals to the original one. See Unser (1999) for further details.

Normalized Kernel Functions. A different approach is to use (a-priori) normalized kernel functions, as for example in Ormoneit and Sen (2002). That is,

$$\phi_i(x) = \frac{\varphi_i(d(x, x_i))}{\sum_{j=1}^m \varphi_j(d(x, x_j))}, \quad i = 1, \dots, m,$$

where $\varphi_i : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ are some basic function with finite support, and $\{x_i\}_{i=1}^m$ are the points at which the kernels are centered (or, “placed”). We note that the well-known Cerebellar Model Articulation Controller (CMAC) (Albus, 1975) fits into above framework. The advantage of these kernel functions is in their inherent normalization property. However, the splitting schemes for such kernels are not trivial in general, and will not be discussed here.

To summarize, any kernels that satisfy Definitions 2 and 3 can be used. However, in what follows, we will mostly focus on the spline-type kernels due to the well-established splitting schemes associated with them.

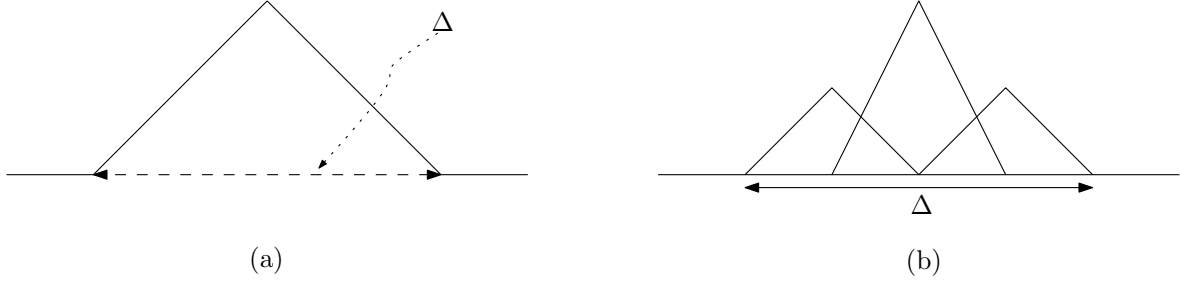


Figure 1: Triangular kernels. (a) Original kernel of support Δ . (b) Resulting kernels after the split.

3.3 Continuity Assumption

The following continuity property will be assumed to hold for the basic model parameters.

Assumption 2. *There exist constants $\alpha > 0$ and $\beta > 0$ such that, for all $x_1, x_2 \in \mathbb{X}$ and $a \in \mathbb{A}$,*

$$|r(x_1, a) - r(x_2, a)| \leq \alpha \cdot d(x_1, x_2), \quad (6a)$$

$$d(f(x_1, a) - f(x_2, a)) \leq \beta \cdot d(x_1, x_2). \quad (6b)$$

A continuity assumption of some kind is obviously essential for generalization in continuous state spaces. Assumptions of similar nature to the one above were used in various works on state aggregation, such as Whitt (1978); Chow and Tsitsiklis (1991). However, we note that the specific assumptions used in these papers refer to continuity of probability densities. Consequently they are too strong for the continuous *deterministic* case as they imply that all states are mapped to the same target state.

We assume that both α and β are known and given to the learning algorithm.

Continuity of the basic model parameters implies continuity of the optimal value function. Specifically, we define the *modulus of continuity* of the optimal value function, and a modulus of continuity bound as follows.

Definition 4. *The modulus of continuity of the optimal value function V is defined as*

$$\omega(z) \triangleq \sup_{x_1, x_2: d(x_1, x_2) \leq z} |V(x_1) - V(x_2)|, \quad z > 0.$$

Definition 5. *Modulus of continuity bound (MCB) is a function $\bar{\omega}(z)$ which satisfies:*

1. $\omega(z) \leq \bar{\omega}(z), \quad \forall z > 0,$
2. $\bar{\omega}(z) \rightarrow 0, \text{ as } z \rightarrow 0.$

Remarks.

1. It is shown in Lemma 1 of Section 7 that the following function satisfies the conditions of Definition 5:

(a) If $\gamma\beta < 1$ (non-expansive case),

$$\bar{\omega}(z) = \frac{\alpha}{1 - \gamma\beta} z. \quad (7)$$

(b) If $\gamma\beta > 1$ (expansive case),

$$\bar{\omega}(z) = cz^{\log_\beta(1/\gamma)}, \quad (8)$$

where

$$c \triangleq 2\beta \left(\frac{\alpha}{\gamma\beta - 1} \right)^{\log_\beta(1/\gamma)} V_b^{\log_\beta(\gamma\beta)} \quad (9)$$

Here α, β are taken from Assumption 2, γ is the discount factor, and V_b is defined in (1).

2. Note that there is a substantial difference in the nature of bounds (7) and (8). This difference can be understood by observing the bound on the distance between optimal values of two states, presented in proof of Lemma 1 of Section 7:

$$|V(x_1) - V(x_2)| \leq \alpha d(x_1, x_2) \sum_{k=0}^{H-1} (\gamma\beta)^k + \gamma^H V_b, \quad H > 0. \quad (10)$$

If $\gamma\beta > 1$, instead of bounding the infinite sum of distances between future rewards, we have to employ a “cut-off tactics”. Specifically, we have to make a balance between the first term in (10), which grows exponentially in H , and the second term, which decays exponentially in H . This results in a more complex bound $\bar{\omega}(z)$, with a worse dependence on z .

3. We assume that a fixed bound $\bar{\omega}$ that satisfies Definition 5 is used throughout.

4 The ARL Algorithm

In this section we present the Adaptive-resolution Reinforcement Learning (ARL) algorithm, which is directly based on the principle of refinement of frequently visited areas of the state-space. In the following subsections we present the different components of this algorithm in detail. An outline of the complete algorithm is presented as Algorithm 1.

4.1 Action Kernel Sets

Our algorithm will employ a separate kernel set for every action. This will allow the use of different resolutions for the different actions. We denote by

$$\Phi_t(a) \triangleq \{\phi_i^a(x)\}_{i=1}^{m_a}$$

the kernel set that is used by the algorithm at time t for action a . Also, for each $a \in \mathbb{A}$ and $\phi \in \Phi_t(a)$, we let

$$N_t(\phi) \triangleq \sum_{\tau=0}^{t-1} \mathbb{I}\{(x_\tau, a_\tau) = (x, a) : x \in \text{supp}(\phi)\}$$

Algorithm 1 ARL Algorithm (outline)

Input parameters:

Maximal reward r_{max} ,
Lipschitz continuity parameters α and β ,
Count threshold \mathcal{N} ,
Size threshold Δ_ϵ .

Initialization:

1. Initialize the kernel set to some feasible kernel set $\Phi_0(a) = \Phi_0$ for all $a \in \mathbb{A}$, and the cell count $N(\phi) = 0$, for all $\phi \in \Phi_0(a)$;
2. For all $a \in \mathbb{A}$ and $\phi \in \Phi_0(a)$, initialize the reward upper uncertainty bound and the transition uncertainty set:

$$\tilde{r}(\phi) = r_{max}, \quad \text{UI}_f(\phi) = \mathbb{X}.$$

For times $t = 0, 1, 2, \dots$ do:

1. **Policy Computation:** Algorithm 2
 2. **Policy Execution:** Algorithm 3
 3. **Kernel Splitting:** Algorithm 4
-

denote the number of times (up to time t) the process visited a state in the support of ϕ and the action a was taken. In our presentation we will frequently refer to this latter event using the terminology “a kernel $\phi \in \Phi_t(a)$ is visited” or “a kernel $\phi \in \Phi_t(a)$ was encountered”.

In what follows, we often drop the time index from our notation for ease of exposition.

4.2 Empirical Model

We use a *single sample* to estimate empirically the reward and transition at the support of each kernel. Specifically, suppose that we choose action a in state x . We thus obtain the sample $(x, a, r = r(x, a), x' = f(x, a))$. Now, for each $\phi \in \Phi_t(a)$ having $x \in \text{supp}(\phi)$, we define the empirical model based on this single sample:

$$\hat{r}(\phi) = r, \tag{11}$$

$$\hat{f}(\phi) = x'. \tag{12}$$

There are several options how to maintain the model in the course of learning:

1. Once the sample for ϕ is obtained, the model remains unchanged for this kernel (until the kernel is split).

2. A more effective strategy is to replace the model of a kernel ϕ using a *better* sample (according to ϕ). For instance, if the current model is based on the sample

$$(x_1, a, r_1 = r(x_1, a), x'_1 = f(x_1, a)),$$

and a new sample

$$(x_2, a, r_2 = r(x_2, a), x'_2 = f(x_2, a))$$

arrives, we will update the model using the new sample if $\phi(x_2) > \phi(x_1)$.

3. Moreover, we can save *all* the samples obtained for ϕ (even if we do not use them for the model). When ϕ will be split, these samples should allow us to initialize the model of the new kernels.

We note that the algorithm and its analysis remain unchanged, regardless of the choice above. For ease of exposition, we assume throughout that the first option is used.

4.3 Confidence Intervals and Upper Value Function

In our algorithm we will use an uncertainty intervals exploration technique as it applies to deterministic systems due to finite approximation.

Recall that $\Delta(\phi)$ denotes the diameter of the support of a kernel function ϕ (equation (5)). At any time t , and for every $a \in \mathbb{A}$ and $\phi \in \Phi_t(a)$, we define the reward uncertainty interval around the empirical reward (11) as:

$$\text{UI}_r(\phi) \triangleq [\hat{r}(\phi) - \alpha\Delta(\phi), \hat{r}(\phi) + \alpha\Delta(\phi)]$$

if the pair (ϕ, a) was sampled till time t ; otherwise, the reward uncertainty interval for this kernel is inherited from the parent kernel. By the continuity Assumption 2, this uncertainty interval satisfies that $r(x, a) \in \text{UI}_r(\phi)$ for all $x \in \text{supp}(\phi)$. In our algorithm only the upper bound of $\text{UI}_r(\phi)$ will be used, which we denote by

$$\tilde{r}(\phi) \triangleq \hat{r}(\phi) + \alpha\Delta(\phi).$$

Next, the transition uncertainty set is defined as:

$$\text{UI}_f(\phi) \triangleq \left\{ y \in \mathbb{X} : d(y, \hat{f}(\phi)) \leq \beta\Delta(\phi) \right\}.$$

If the pair (ϕ, a) was not sampled till time t , the uncertainty set is inherited from the parent kernel as in the reward case. Again, by the continuity assumption, this uncertainty set satisfies: $f(x, a) \in \text{UI}_f(\phi)$ for all $x \in \text{supp}(\phi)$. Using this notation, we define the following dynamic programming (DP) operator.

Definition 6. *The upper DP operator at time t is defined for any given function $g : \mathbb{X} \rightarrow \mathbb{R}$ by*

$$\mathcal{T}g(x) = \max_{a \in \mathbb{A}} \left\{ \sum_{\phi \in \Phi_t(a)} \phi(x) \left(\tilde{r}(\phi) + \gamma \left[\min_{y \in \text{UI}_f(\phi)} g(y) + \bar{\omega}(2\beta\Delta(\phi)) \right] \right) \right\},$$

where $\bar{\omega}$ is an MCB (Definition 5; see also Remark 1 after that definition).

Now, using this operator, we define *the upper value function* (UVF) as the solution of the following fixed point equation:

$$\tilde{V}_t(x) = \mathcal{T}\tilde{V}_t(x), \quad x \in \mathbb{X}. \quad (13)$$

It can be verified that this equation has a unique solution (Lemma 2 in Section 7). Note however, that since this solution is defined on an infinite state space \mathbb{X} , its computation is not a trivial issue in general. We will discuss the computational aspects in Section 5.

We will show that the UVF is indeed an upper bound on the optimal value function V (Lemma 3 in Section 7). In fact, the operator \mathcal{T} can be interpreted as an upper-bound approximation of the optimal Bellman's operator (cf. equation (2)) using a basis of kernel functions. Indeed, for $\phi \in \Phi_t(a)$, the term

$$\tilde{Q}_t(\phi) \triangleq \tilde{r}(\phi) + \gamma \left[\min_{y \in \text{UI}_f(\phi)} \tilde{V}_t(y) + \bar{\omega}(2\beta\Delta(\phi)) \right] \quad (14)$$

is an upper-bound approximation of the optimal Q -function $Q(x, a)$ for all x in the support of ϕ : $\tilde{r}(\phi)$ is an upper bound on the immediate reward that can be obtained for such x using a , while $\min_{y \in \text{UI}_f(\phi)} \tilde{V}_t(y) + \bar{\omega}(2\beta\Delta(\phi))$ is an upper bound on the next state value. Note that the chosen upper bound on the next state value is the *tightest* possible one among those in the transition uncertainty set of ϕ .

In addition to providing an upper bound for V , we will show that the UVF converges to V as the kernel supports shrink (Lemma 4 in Section 7).

The policy that is used in the algorithm is now the optimal (or *greedy*) policy with respect to $\tilde{V}_t(x)$:

$$\pi_t(x) = \operatorname{argmax}_{a \in \mathbb{A}} \left\{ \sum_{\phi \in \Phi_t(a)} \phi(x) \tilde{Q}_t(\phi) \right\}, \quad x \in \mathbb{X}.$$

We summarize the UVF and policy computation algorithm in Algorithm 2, and the policy execution process in Algorithm 3.

4.4 Splitting Method

Assume that a fixed uniform splitting scheme is used throughout (cf. Definition 3). Define a count threshold \mathcal{N} . We will split a kernel if the number of visits in its support exceeds \mathcal{N} . Now, let $\phi \in \Phi_t(a)$ be a parent kernel which is split at time t into K new kernels $\phi_1, \dots, \phi_K \in \Phi_{t+1}(a)$, and denote by $(x, a, \hat{r}(\phi), \hat{f}(\phi))$ the sample of the parent kernel, with $x \in \text{supp}(\phi)$, $\hat{r}(\phi) = r(x, a)$, $\hat{f}(\phi) = f(x, a)$. We initialize the model of the new kernels $\{\phi_j\}$ as follows:

1. If $x \in \text{supp}(\phi_j)$, let ϕ_j inherit the sample of the parent kernel. Namely, set

$$\begin{aligned} \tilde{r}(\phi_j) &= \hat{r}(\phi) + \alpha\Delta(\phi_j), \\ \text{UI}_f(\phi_j) &= \left\{ y \in \mathbb{X} : d(y, \hat{f}(\phi)) \leq \beta\Delta(\phi_j) \right\}. \end{aligned}$$

2. Otherwise, initialize the uncertainty parameters to those of the parent kernel:

$$\begin{aligned} \tilde{r}(\phi_j) &= \tilde{r}(\phi), \\ \text{UI}_f(\phi_j) &= \text{UI}_f(\phi). \end{aligned}$$

Algorithm 2 Policy Computation

If the model has changed (that is, some kernel has been visited for the first time, or some kernel has been split):

1. Compute the upper value function (UVF) by solving

$$\tilde{V}_t(x) = \mathcal{T}\tilde{V}_t(x), x \in \mathbb{X}, \quad (15)$$

where \mathcal{T} is specified in Definition 6.

2. Compute the corresponding optimal policy

$$\pi_t(x) = \operatorname{argmax}_{a \in \mathbb{A}} \left\{ \sum_{\phi \in \Phi_t(a)} \phi(x) \tilde{Q}(\phi) \right\}, x \in \mathbb{X}, \quad (16)$$

where \tilde{Q} is defined in (14). If more than one action achieves the maximum, choose the first one in lexicographic order.

Otherwise, use the previously computed value and policy: $\tilde{V}_t = \tilde{V}_{t-1}$ and $\pi_t = \pi_{t-1}$.

Algorithm 3 Policy Execution

Execute the action $a_t = \pi_t(x_t)$, and observe $r_t = r(x_t, a_t)$, $x_{t+1} = f(x_t, a_t)$. For every $\phi \in \Phi_t(a_t)$ such that $x_t \in \operatorname{supp}(\phi)$:

- (i) Update the visits counter: $N(\phi) := N(\phi) + 1$.
- (ii) If ϕ is visited *for the first time*, compute the model for this kernel. Namely,
 - (a) Compute the empirical reward and transition according to equations (11) and (12).
 - (b) Compute the upper reward value

$$\tilde{r}(\phi) := \hat{r}(\phi) + \alpha \Delta(\phi), \quad (17)$$

- (c) Compute the transition uncertainty set

$$\operatorname{UI}_f(\phi) := \left\{ y \in \mathbb{X} : d(y, \hat{f}(\phi)) \leq \beta \Delta(\phi) \right\}. \quad (18)$$

We note that this choice of the initialization leads to an improvement of the uncertainty parameters of the new kernels which contain the sample of the parent kernel. Moreover, if the whole history of samples is saved in the course of learning, we can further improve step (2) above by using corresponding samples to initialize the model of the new kernels.

In addition to this splitting criterion, we also employ a “stop-splitting” rule, based on the kernel support diameter. Let Δ_ϵ be a (small) *size threshold* parameter. Then, if a kernel ϕ satisfies $\Delta(\phi) \leq \Delta_\epsilon$, it will not be split anymore. Since the number of times that the algorithm encounters a kernel with $\Delta(\phi) > \Delta_\epsilon$ can be bounded, it follows that the number of different stationary policies that the algorithm computes and uses can also be bounded. This will eventually enable us to prove a bound on the policy-mistake count, in Section 7.

Now, under a fixed uniform splitting scheme, we denote by Φ_ϵ the coarsest achievable kernel set with $\Delta(\phi) \leq \Delta_\epsilon$ for all $\phi \in \Phi_\epsilon$. We call this set Δ_ϵ -supported kernel set. The number of kernels in Φ_ϵ can be bounded as follows (see Lemma 6 in Section 7):

$$N_\epsilon \triangleq |\Phi_\epsilon| \leq |\Phi_0| K \left(\frac{\Delta_{max}}{\Delta_\epsilon} \right)^{\log_{1/\lambda}(K)}, \quad (19)$$

where Φ_0 is the initial kernel set, λ and K are the parameters of the splitting scheme (Definition 3), and Δ_{max} is the diameter of the state space (from Assumption 1).

We summarize the splitting process in Algorithm 4. This completes the description of the ARL algorithm as outlined in Algorithm 1.

5 UVF Computation Procedure

In this section we propose a general procedure for solving the fixed point equation (13). Recall the definition of the upper value DP operator \mathcal{T} (Definition 6). In Lemma 2 of Section 7 we show that this operator is a contraction mapping and, in principle, the solution \tilde{V}_t of the fixed point equation (13) can be found using Value Iteration. Now, since this solution is defined on the infinite state space \mathbb{X} , a straightforward application of Value Iteration is infeasible. However, a more careful examination of Definition 6 reveals that in fact only the following *finite* set of *target values* is needed in order to compute the value \tilde{V}_t and the policy π_t at any $x \in \mathbb{X}$:

$$\tilde{T}(i, a) \triangleq \min_{y \in \text{UI}_f(\phi_i^a)} \tilde{V}_t(y), \quad a \in \mathbb{A}, \quad \phi_i^a \in \Phi_t(a), \quad i = 1, \dots, |\Phi_t(a)|. \quad (20)$$

Indeed, we can rewrite equation (13) in terms of $\{\tilde{T}(i, a)\}$ as follows:

$$\tilde{V}_t(x) = \max_{a \in \mathbb{A}} \left\{ \sum_{i=1}^{|\Phi_t(a)|} \phi_i^a(x) \left(\tilde{r}(\phi_i^a) + \gamma [\tilde{T}(i, a) + \bar{\omega}(i, a)] \right) \right\},$$

where

$$\bar{\omega}(i, a) \triangleq \bar{\omega}(2\beta\Delta(\phi_i^a))$$

This implies that

$$\tilde{T}(i, a) = \min_{x \in \text{UI}_f(\phi_i^a)} \max_{a' \in \mathbb{A}} \left\{ \sum_{j=1}^{|\Phi_t(a')|} \phi_j^{a'}(x) \left(\tilde{r}(\phi_j^{a'}) + \gamma [\tilde{T}(j, a') + \bar{\omega}(j, a')] \right) \right\}, \quad (21)$$

Algorithm 4 Splitting Algorithm

1. Initialize $\Phi_{t+1}(a) = \Phi_t(a)$, for all $a \in \mathbb{A}$.
2. For each $a \in \mathbb{A}$ and $\phi \in \Phi_t(a)$ which satisfy $N(\phi) \geq \mathcal{N}$ and $\Delta(\phi) > \Delta_\epsilon$, perform the following:
 - (a) Split this kernel according to the given uniform splitting scheme. Let $\phi_1, \dots, \phi_K \in \Phi_{t+1}(a)$ be the resulting sub-kernels after this split.
 - (b) Initialize the reward upper uncertainty bounds of the kernels:

$$\tilde{r}(\phi_j) = \begin{cases} \hat{r}(\phi) + \alpha \Delta(\phi_j), & \text{if } \phi_j \text{ contains the sample of the parent kernel } \phi, \\ \tilde{r}(\phi), & \text{otherwise.} \end{cases}$$

- (c) Initialize the transition uncertainty sets of the new kernels:

$$\text{UI}_f(\phi_j) = \begin{cases} \left\{ y \in \mathbb{X} : d(y, \hat{f}(\phi)) \leq \beta \Delta(\phi_j) \right\}, & \text{if } \phi_j \text{ contains the sample of the parent kernel } \phi, \\ \text{UI}_f(\phi), & \text{otherwise.} \end{cases}$$

- (d) Update the counts of the new cells as follows:

$$N(\phi_j) = \begin{cases} 1, & \text{if } \phi_j \text{ contains the sample of the parent kernel } \phi, \\ 0, & \text{otherwise.} \end{cases}$$

for all $a \in \mathbb{A}$, $\phi_i^a \in \Phi_t(a)$, $i = 1, \dots, |\Phi_t(a)|$.

Finally, we can show (using the same arguments as in proof of Lemma 2) that the operator associated with equation (21) is a contraction mapping (with contraction factor γ), so that the target values (20) can be computed using the following Value Iteration procedure:

1. Initialize $\tilde{T}^{(0)}(i, a) \equiv V_{\max}$.
2. At iteration $k = 1, 2, \dots$, update:

$$\tilde{T}^{(k)}(i, a) = \min_{x \in \text{UI}_f(\phi_i^a)} \max_{a' \in \mathbb{A}} \left\{ \sum_{j=1}^{|\Phi_t(a')|} \phi_j^{a'}(x) \left(\tilde{r}(\phi_j^{a'}) + \gamma \left[\tilde{T}^{(k-1)}(j, a') + \bar{\omega}(j, a') \right] \right) \right\}. \quad (22)$$

Unfortunately, in order to carry out the above procedure, one in principle must solve an infinite number of minmax problems⁴ (22) of a not necessarily convex function. This is of course infeasible in general. However, at least in the special cases of constant and triangular kernels, this procedure can be practically implemented. See Appendix for details.

⁴In fact, in Section 8 we will present an incremental variant of our algorithm, in which only few iterations are performed at each time step. This however still leaves us with the need of solving the minmax problem above.

6 Main Results

We next summarize the main results regarding our algorithm. Proofs are deferred to the next section.

Recall the definition of the policy-mistake count (4) and the corresponding near-optimality criterion. Also, recall that Φ_ϵ is the coarsest achievable kernel set with $\Delta(\phi) \leq \Delta_\epsilon$, and satisfies (19). First we present the main theorem, which provides a mistake bound of our scheme in terms of the number of kernels in Φ_ϵ .

Theorem 1. *Let $\epsilon > 0$ be given and assume that the ARL algorithm's input parameter Δ_ϵ is chosen such that*

$$\frac{2\alpha\Delta_\epsilon + \gamma\bar{\omega}(2\beta\Delta_\epsilon)}{1 - \gamma} \leq \frac{\epsilon}{2}. \quad (23)$$

Then, the policy-mistake count of the algorithm is bounded by

$$\text{PMC}(\epsilon) \leq \frac{|\Phi_\epsilon| |\mathbb{A}| (\mathcal{N} + 1)}{1 - \gamma} \ln \frac{2V_b}{\epsilon}.$$

Remark. As shown in Lemma 5, an explicit expression for Δ_ϵ , which ensures that (23) is satisfied, can be obtained when using the MCB $\bar{\omega}$ defined in (7) and (8), as follows:

1. If $\gamma\beta < 1$, let

$$\Delta_\epsilon = \frac{(1 - \gamma)(1 - \gamma\beta)}{4\alpha} \epsilon. \quad (24)$$

2. If $\gamma\beta > 1$, let

$$\Delta_\epsilon = \left(\frac{(1 - \gamma)\epsilon}{2(2\alpha + \gamma c (2\beta)^{\log_\beta(1/\gamma)})} \right)^{\log_{1/\gamma}\beta}, \quad (25)$$

where c is defined in (9).

Using the above formulas and (19) one can obtain an explicit dependence of the mistake bound on ϵ (and other parameters). See the ‘‘Discussion’’ subsection below for a further elaboration on the nature of the bounds for $\gamma\beta < 1$ and $\gamma\beta > 1$.

As a corollary of this theorem, we have the following.

Theorem 2. *Consider the non-adaptive learning algorithm which uses the Δ_ϵ -supported kernel set $\Phi_t(a) = \Phi_\epsilon$ for all $t \geq 0$ and $a \in \mathbb{A}$, with Δ_ϵ satisfying (23). Then, the policy-mistake count of the non-adaptive algorithm is bounded by*

$$\text{PMC}(\epsilon) \leq \frac{|\Phi_\epsilon| |\mathbb{A}|}{1 - \gamma} \ln \frac{2V_b}{\epsilon}.$$

Proof. Follows immediately from Theorem 1, by considering the ARL algorithm (Algorithm 1) initialized to $\Phi_0(a) \equiv \Phi_\epsilon$ and using $\mathcal{N} = 0$. \square

In addition to Theorem 1, we can obtain a possibly tighter mistake bound in terms of the *posterior number of kernels actually used in the course of the algorithm*. In fact, the purpose of adaptive-resolution approximation is that as time progresses, the algorithm will refine kernels only in the vicinity of the optimal trajectory. Therefore, the actual number of kernels in the long term should be much less than $|\Phi_\epsilon|$. We make this more formal below.

Definition 7. Let x_0 be the initial state and let $N_\infty(x_0, a)$ be the number of cells in the grid $\Phi_t(a)$ as $t \rightarrow \infty$, that is

$$N_\infty(x_0, a) \triangleq \lim_{t \rightarrow \infty} |\Phi_t(a)|. \quad (26)$$

Also, let $N_\infty(x_0) \triangleq \sum_{a \in \mathbb{A}} N_\infty(x_0, a)$.

We note that the limit in (26) exists and is finite, since $|\Phi_t(a)|$ increases in t , while $|\Phi_t(a)| \leq |\Phi_\epsilon|$ due to the enforced “stop-splitting” rule. For the same reason, we have the trivial bound $N_\infty(x_0) \leq |\Phi_\epsilon| |\mathbb{A}|$.

Theorem 3. Let $\epsilon > 0$ and assume that the ARL algorithm receives an input parameter Δ_ϵ as in Theorem 1. Then, it holds that

$$\text{PMC}(\epsilon) \leq \frac{2N_\infty(x_0)\mathcal{N}}{1-\gamma} \ln \frac{2V_b}{\epsilon}.$$

Remark. The bound of Theorem 3 becomes better than that of Theorem 1 if $N_\infty(x_0) \leq \frac{1}{2} |\Phi_\epsilon| |\mathbb{A}|$.

Discussion

First, consider the contractive case ($\gamma\beta < 1$). Theorem 1 implies that the mistake bound is linear in $|\Phi_\epsilon|$. Therefore, using (19) and (24), we obtain the following explicit dependence on ϵ (ignoring the log factor):

$$\text{PMC}(\epsilon) \lesssim C (1/\epsilon)^n |\mathbb{A}| (\mathcal{N} + 1), \quad (27)$$

where the constant C is polynomial in α , $1/(1-\gamma)$ and in $1/(1-\gamma\beta)$. Note that the exponential dependence on the dimension n of the state-space is an obvious artifact of the dense aggregation approach, since in most cases $\log_{1/\lambda}(K)$, which appears in (19), is of order of n .

In the expansive case ($\gamma\beta > 1$), the analysis becomes more involved, as explained in Remark 2 after Definition 5. In particular, it is shown in Section 7 that to obtain the mistake bounds of Theorems 1 and 3, the size threshold should be taken as in equation (25). As a result, in the expansive case we obtain a worse explicit dependence of the mistake bound on ϵ , as follows:

$$\text{PMC}(\epsilon) \lesssim C' (1/\epsilon)^{n \log_{1/\gamma} \beta} |\mathbb{A}| (\mathcal{N} + 1),$$

where C' is polynomial in α , β , $1/(1-\gamma)$, and exponential in $\log_{1/\gamma} \beta$. Note that in this case $\log_{1/\gamma} \beta > 1$.

In the context of the posterior bound (Theorem 3), it should be noted that there is a trade-off between the choice of the count threshold \mathcal{N} and the number of kernels at infinity $N_\infty(x_0)$. If we choose \mathcal{N} too small, the algorithm will perform many splits, and consequently $N_\infty(x_0)$ will be large. In this case it may happen that the algorithm will produce redundant kernels, which are not actually needed for near-optimal performance. On the other hand, if we choose large \mathcal{N} , the algorithm will perform less splits, resulting in a smaller $N_\infty(x_0)$. This however may lead to a slower convergence to the optimal trajectory.

Two comparisons that may be of interest follow. First, consider our algorithm for the “flat” model, which uses a sufficiently fine kernel set (namely, Φ_ϵ) over the state space. As Theorem 2 shows, the mistake bound in such case will be as in Theorem 1, with $\mathcal{N} + 1$ replaced by 1. Clearly, however, such an algorithm is not feasible if $|\Phi_\epsilon|$ is large.

Moreover, consider a naïve approach, where the “flat” model is treated as a finite-state MDP, and an efficient exploration technique is used on this MDP (such as the R-MAX algorithm by Brafman and Tennenholtz (2002)). In the ideal case when the MDP assumption happens to hold true, such an algorithm will again have the mistake bound as in Theorem 2 (see for instance Kakade (2003), Theorem 8.3.5). However, as this assumption generally is not satisfied, the computed value function might *underestimate* the optimal one, resulting in algorithm’s failure to converge to the ϵ -optimal policy (and, in fact, in infinite mistake count).

7 Analysis of the ARL Algorithm

Below is the outline of the analysis. First we show that there exists an MCB (Definition 5), which will justify Definition 6. Then, we show that there exists a unique solution to equation (13), and that this solution upper bounds the optimal value V . Finally, we prove that under certain conditions on the kernel sets, the optimal policy with respect to the UVF (equation (16)) is an ϵ -optimal policy, which will enable us to prove a polynomial bound on the policy-mistake count of the algorithm.

To alleviate the notation, throughout the analysis we write UI for the transition uncertainty set (instead of UI_f).

7.1 Continuity of the Optimal Value Function

In this subsection we show that under continuity Assumption 2, the optimal value function is also uniformly continuous. However, it will be Lipschitz continuous, only when $\gamma\beta < 1$. Otherwise, a weaker sort of continuity, namely Hölder continuity, holds. This is shown by the following lemma.

Lemma 1. *For any given $x_1, x_2 \in \mathbb{X}$, we have that*

$$|V(x_1) - V(x_2)| \leq \bar{\omega}(d(x_1, x_2)),$$

where $\bar{\omega}$ is defined in (7) and (8).

Proof. Fix $x_1, x_2 \in \mathbb{X}$. From the optimality equation (2), we have that

$$\begin{aligned} |V(x_1) - V(x_2)| &\leq \max_a |r(x_1, a) - r(x_2, a)| + \gamma \max_a |V(f(x_1, a)) - V(f(x_2, a))| \\ &\leq \alpha d(x_1, x_2) + \gamma \max_a |V(f(x_1, a)) - V(f(x_2, a))|, \end{aligned}$$

where the second inequality follows by Assumption 2. Also by this assumption, we have that

$$d(f(x_1, a), f(x_2, a)) \leq \beta d(x_1, x_2),$$

for any a . Applying the above inequalities iteratively, for any integer $H > 0$, we obtain the following bound

$$|V(x_1) - V(x_2)| \leq \alpha d(x_1, x_2) \sum_{k=0}^{H-1} (\gamma\beta)^k + \gamma^H V_b.$$

First, consider the case of $\gamma\beta < 1$. Here we can take $H = \infty$ in the above bound, and obtain the desired result. Next, if $\gamma\beta > 1$, we have that

$$\begin{aligned} |V(x_1) - V(x_2)| &\leq \alpha d(x_1, x_2) \frac{(\gamma\beta)^H - 1}{\gamma\beta - 1} + \gamma^H V_b \\ &\leq \alpha d(x_1, x_2) \frac{(\gamma\beta)^H}{\gamma\beta - 1} + \gamma^H V_b. \end{aligned}$$

Now, since $\gamma < 1$, there exist a *minimal* $H = H_0(x_1, x_2)$, such that

$$\gamma^{H_0} V_b \leq \alpha d(x_1, x_2) \frac{(\gamma\beta)^{H_0}}{\gamma\beta - 1}, \quad (28)$$

implying that

$$|V(x_1) - V(x_2)| \leq 2\alpha d(x_1, x_2) \frac{(\gamma\beta)^{H_0}}{\gamma\beta - 1}. \quad (29)$$

Since H_0 is the minimal H satisfying (28), it holds that

$$\alpha d(x_1, x_2) \frac{\beta^{H_0-1}}{\gamma\beta - 1} < V_b \leq \alpha d(x_1, x_2) \frac{\beta^{H_0}}{\gamma\beta - 1},$$

implying that $\beta^{H_0} < \frac{V_b \beta (\gamma\beta - 1)}{\alpha d(x_1, x_2)}$, $\beta^{H_0} \geq \frac{V_b (\gamma\beta - 1)}{\alpha d(x_1, x_2)}$, and consequently,

$$\gamma^{H_0} \leq \left(\frac{V_b (\gamma\beta - 1)}{\alpha d(x_1, x_2)} \right)^{\log_{\beta} \gamma}.$$

Therefore, we have that

$$(\gamma\beta)^{H_0} \leq \beta \left(\frac{V_b (\gamma\beta - 1)}{\alpha d(x_1, x_2)} \right)^{\log_{\beta} \gamma\beta} \quad (30)$$

Substituting (30) in (29) yields the result. \square

7.2 The Upper Value Function

Below we show that the operator \mathcal{T} (Definition 6) is a contraction operator, and that the solution of equation (13) is indeed an upper bound on the optimal value function.

Lemma 2. *The operator \mathcal{T} is a contraction mapping in the ℓ_{∞} norm, with the contraction factor γ . Thus, there exists a unique solution to equation (13).*

Proof. Given two functions g_1 and g_2 , we have the following sequence of inequalities for any $x \in \mathbb{X}$:

$$\begin{aligned}
|(\mathcal{T}g_1)(x) - (\mathcal{T}g_2)(x)| &\leq \gamma \max_{a \in \mathbb{A}} \left| \sum_{\phi \in \Phi_t(a)} \phi(x) \left(\min_{y \in \text{UI}(\phi)} g_1(y) - \min_{y \in \text{UI}(\phi)} g_2(y) \right) \right| \\
&\leq \gamma \max_{a \in \mathbb{A}} \sum_{\phi \in \Phi_t(a)} \phi(x) \max_{y \in \text{UI}(\phi)} |g_1(y) - g_2(y)| \\
&\leq \gamma \max_{y \in \mathbb{X}} |g_1(y) - g_2(y)| \sum_{\phi \in \Phi_t(a)} \phi(x) \\
&= \gamma \max_{y \in \mathbb{X}} |g_1(y) - g_2(y)| \triangleq \|g_1 - g_2\|_\infty,
\end{aligned}$$

where the first inequality follows by Definition 6 and the equality follows by the definition of kernel set (Definition 2). Hence, $\|\mathcal{T}g_1 - \mathcal{T}g_2\|_\infty \leq \gamma \|g_1 - g_2\|_\infty$, which proves the result. \square

Lemma 3. *The UVF \tilde{V}_t is indeed an upper bound on the optimal value function. That is, at every time t , we have that*

$$\tilde{V}_t(x) \geq V(x), \quad \forall x \in \mathbb{X}.$$

Proof. Since, by Lemma 2, \mathcal{T} is a contraction operator, we can prove the claim by induction on the steps of value iteration. For the base case, let $\tilde{V}^0(x) = V_{\max} \geq V(x), \forall x \in \mathbb{X}$. Now assume that the claim holds for n -th iteration. For $n + 1$ -th iteration we have by the Lipschitz continuity of the reward (Assumption 2) and by the definition of $\tilde{r}(\phi)$, that for all $a \in \mathbb{A}$, $\phi \in \Phi(a)$, and $x \in \text{supp}(\phi)$

$$\tilde{r}(\phi) = r(x_s, a) + \alpha \Delta(\phi) \geq r(x, a),$$

where x_s is a sample point of ϕ . Therefore, by the fact that $\sum_{\phi \in \Phi(a)} \phi(x) = 1$ for all x (Definition 2),

$$\sum_{\phi \in \Phi(a)} \phi(x) \tilde{r}(\phi) \geq r(x, a), \quad \forall x \in \mathbb{X}, a \in \mathbb{A}. \quad (31)$$

Now, set $y^* = \min_{y \in \text{UI}(\phi)} \tilde{V}^n(y)$. For any $x \in \text{supp}(\phi)$, by Assumption 2 and by the definition of $\text{UI}(\phi)$, we have that $f(x, a) \in \text{UI}(\phi)$, implying that

$$d(y^*, f(x, a)) \leq 2\beta \Delta(\phi).$$

Therefore, by the definition of MCB (Definition 5), $|V(y^*) - V(f(x, a))| \leq \bar{\omega}(2\beta \Delta(\phi))$. Consequently,

$$\tilde{V}^n(y^*) + \bar{\omega}(2\beta \Delta(\phi)) \geq V(y^*) + \bar{\omega}(2\beta \Delta(\phi)) \geq V(f(x, a)),$$

where the first inequality follows by the induction assumption. We thus have shown that

$$\sum_{\phi \in \Phi(a)} \phi(x) \left(\min_{y \in \text{UI}(\phi)} \tilde{V}^n(y) + \bar{\omega}(2\beta \Delta(\phi)) \right) \geq V(f(x, a)), \quad \forall x \in \mathbb{X}, a \in \mathbb{A}. \quad (32)$$

Combining (31) and (32) we obtain that

$$\begin{aligned}
\tilde{V}^{n+1}(x) &= \max_{a \in \mathbb{A}} \left\{ \sum_{\phi \in \Phi(a)} \phi(x) \tilde{r}(\phi) + \gamma \sum_{\phi \in \Phi(a)} \phi(x) \left(\min_{y \in \text{UI}(\phi)} \tilde{V}^n(y) + \bar{\omega}(2\beta\Delta(\phi)) \right) \right\} \\
&\geq \max_{a \in \mathbb{A}} \{r(x, a) + \gamma V(f(x, a))\} \\
&= V(x),
\end{aligned}$$

which completes the induction proof. Since $\tilde{V}^n \rightarrow \tilde{V}$, the result follows. \square

7.3 Near-Optimality of the UVF Optimal Policy

In this section we provide a sufficient condition on the diameter of the support of kernels, which ensures that the return obtained by the policy $\mathcal{A}_t = \{\pi_\tau\}_{\tau=t}^\infty$ which the algorithm implements at time t , is ϵ -close to the UVF: $\tilde{V}_t(x) - J_M^{\mathcal{A}_t}(x) \leq \epsilon$. This will imply that $V(x) - J_M^{\mathcal{A}_t}(x) \leq \epsilon$, since \tilde{V}_t is an upper bound on the optimal value; namely, this will imply that \mathcal{A}_t is an ϵ -optimal policy from x .

The following is a standard definition, which specifies a mixing time of any stationary policy in discounted MDPs.

Definition 8 ($\epsilon/2$ -Horizon Time). *In an MDP M , the $\epsilon/2$ -horizon time is given by*

$$T_{\epsilon/2} \triangleq \log_{1/\gamma} \frac{2V_b}{\epsilon}.$$

To proceed, we introduce the definitions of *known kernel-action pairs* and the *escape event*.

Definition 9 (Known Pairs). *At any time t , define the set of known kernel-action pairs:*

$$\mathcal{K}_t \triangleq \{(\phi, a) \in \Phi_t(a) \times \mathbb{A} : \Delta(\phi) \leq \Delta_\epsilon, \phi \text{ was previously sampled}\}.$$

Definition 10 (Escape Event). *At any time t , define the escape event from a given starting state $x \in \mathbb{X}$:*

$$E_t(x) \triangleq \left\{ \text{In a trial generated by starting from state } x \text{ and following } \mathcal{A}_t \text{ for } T_{\epsilon/2} \text{ steps in } M, \text{ a pair } (x_t, a_t) \text{ is encountered, such that } x_t \in \text{supp}(\phi), \text{ but } (\phi, a_t) \notin \mathcal{K}_t \right\}.$$

Definition 11 (Episode). *An episode is a maximal period of time $[t_0, t_1]$, in which:*

(i) *All visited state-action pairs (x_t, a_t) at times $t = t_0, \dots, t_1 - 1$ satisfy that for each kernel $\phi \in \Phi_t(a_t)$ with $x_t \in \text{supp}(\phi)$, it holds that*

$$(a) \quad \Delta(\phi) \leq \Delta_\epsilon,$$

(b) *ϕ was previously sampled (that is, was previously visited).*

(ii) *At time $t = t_1$, the algorithm encounters a pair for which the condition in (i) is not true.*

Note that during each episode, a fixed stationary policy is used by the algorithm, and the policy is (potentially) changed only at the beginning of each episode. Also, observe that the above definitions depend on the size threshold parameter Δ_ϵ . The next lemma formulates the condition on Δ_ϵ which will imply that the execution of the algorithm's implemented policy \mathcal{A}_t from time t will obtain a return which is ϵ -close to the UVF.

Lemma 4. *Let $\epsilon > 0$ be given and assume that Δ_ϵ is chosen such that*

$$\frac{1}{1-\gamma} \left[2\alpha\Delta_\epsilon + \gamma\bar{\omega}(2\beta\Delta_\epsilon) \right] \leq \frac{\epsilon}{2}. \quad (33)$$

Then,

$$\tilde{V}_t(x) - J_M^{\mathcal{A}_t}(x) \leq \epsilon + \mathbb{I}\{E_t(x)\}V_b$$

holds for all t and $x \in \mathbb{X}$.

Proof. At given time t_0 , we consider the execution of the (non-stationary) policy \mathcal{A}_{t_0} for $T_{\epsilon/2}$ time steps in M . Now, we have two mutually exclusive cases:

- (a) For every (x_t, a_t) it holds that for each kernel $\phi \in \Phi_t(a_t)$ with $x_t \in \text{supp}(\phi)$, $\Delta(\phi) \leq \Delta_\epsilon$ and ϕ was sampled.
- (b) There exists at least one $t \in [t_0, t_0 + T_{\epsilon/2} - 1]$ such that the above condition does not hold.

The case (b) above is easy – if it happens, we have that a pair (ϕ, a) not in \mathcal{K}_{t_0} is encountered. Thus, the escape event $E_{t_0}(x_{t_0})$ occurred during the execution of \mathcal{A}_{t_0} for $T_{\epsilon/2}$ time-steps, which is expressed by the $\mathbb{I}\{E_{t_0}(x_{t_0})\}V_b$ term in the bound.

Now, if (a) is the case, during the execution of \mathcal{A}_{t_0} for $T_{\epsilon/2}$ time steps we stay in the same episode (Definition 11), and thus the algorithm's policy remains unchanged and it is the stationary policy π_{t_0} . For simplicity, assume $t_0 = 0$, write π for π_0 and \tilde{V} for \tilde{V}_0 , and recall that π is the greedy policy with respect to \tilde{V} (equation (16)). Thus,

$$\tilde{V}(x_0) = \sum_{\phi \in \Phi(a_0)} \phi(x_0) \tilde{r}(\phi) + \gamma \sum_{\phi \in \Phi(a_0)} \phi(x_0) \left(\min_{y \in \text{UI}(\phi)} \tilde{V}(y) + \bar{\omega}(2\beta\Delta(\phi)) \right).$$

Also, by Bellman's equation,

$$J_M^\pi(x_0) = r(x_0, a_0) + \gamma J_M^\pi(x_1).$$

Now, we have that

$$\begin{aligned} \tilde{V}(x_0) - J_M^\pi(x_0) &\leq 2\alpha\Delta_\epsilon + \gamma \left(\sum_{\phi \in \Phi(a_0)} \phi(x_0) \left(\min_{y \in \text{UI}(\phi)} \tilde{V}(y) + \bar{\omega}(2\beta\Delta(\phi)) \right) - J_M^\pi(x_1) \right) \end{aligned} \quad (34)$$

$$\begin{aligned} &\leq 2\alpha\Delta_\epsilon + \gamma \left(\left(\tilde{V}(x_1) + \bar{\omega}(2\beta\Delta_\epsilon) \right) \left[\sum_{\phi \in \Phi(a_0)} \phi(x_0) \right] - J_M^\pi(x_1) \right) \quad (35) \\ &\leq 2\alpha\Delta_\epsilon + \gamma\bar{\omega}(2\beta\Delta_\epsilon) + \gamma \left(\tilde{V}(x_1) - J_M^\pi(x_1) \right). \end{aligned}$$

Here, inequality (42) follows by the continuity assumption on the reward, since for every kernel $\phi \in \Phi(a_0)$ with $x_0 \in \text{supp}(\phi)$ we have that

$$\begin{aligned}\tilde{r}(\phi) - r(x_0, a_0) &= r(x', a_0) + \alpha\Delta(\phi) - r(x_0, a_0) \\ &\leq 2\alpha\Delta(\phi) \leq 2\alpha\Delta_\epsilon,\end{aligned}$$

where x' is the sample that was received for this kernel, and the last inequality is due to the fact that we are within an episode. Inequality (42) holds since $x_1 \in \text{UI}(\phi)$ for every $\phi \in \Phi(a_0)$ with $x_0 \in \text{supp}(\phi)$. Thus, proceeding iteratively, we obtain the following bound:

$$\tilde{V}(x_0) - J_M^\pi(x_0) \leq \sum_{t=0}^{T_{\epsilon/2}-1} \gamma^t \left[2\alpha\Delta_\epsilon + \gamma\bar{\omega}(2\beta\Delta_\epsilon) \right] + \gamma^{T_{\epsilon/2}} V_b.$$

By the definition of $T_{\epsilon/2}$ (Definition 8), we have $\gamma^{T_{\epsilon/2}} V_b \leq \frac{\epsilon}{2}$.

Now, the first term above can be trivially bounded as follows:

$$\begin{aligned}\sum_{t=0}^{T_{\epsilon/2}-1} \gamma^t \left[2\alpha\Delta_\epsilon + \gamma\bar{\omega}(2\beta\Delta_\epsilon) \right] &\leq \sum_{t=0}^{\infty} \gamma^t \left[2\alpha\Delta_\epsilon + \gamma\bar{\omega}(2\beta\Delta_\epsilon) \right] \\ &= \frac{1}{1-\gamma} \left[2\alpha\Delta_\epsilon + \gamma\bar{\omega}(2\beta\Delta_\epsilon) \right] \\ &\leq \frac{\epsilon}{2},\end{aligned}$$

where the last inequality follows by the hypothesis (33) of the Lemma. This completes the proof. \square

Condition (33) does not provide an explicit condition on the algorithm's input Δ_ϵ . However, such condition can be obtained, at least in case $\gamma\beta \neq 1$, using the particular choice of MCB $\bar{\omega}$ presented in Lemma 1, which is proved in the following.

Lemma 5. *Suppose that $\gamma\beta \neq 1$. Then, the precondition (33) of Lemma 4 is satisfied under the following assumptions.*

1. *If $\gamma\beta < 1$, let Δ_ϵ satisfy (24).*
2. *If $\gamma\beta > 1$, let Δ_ϵ satisfy (25).*

Proof. If $\gamma\beta < 1$, by the explicit expression of $\bar{\omega}$ (see Lemma 1), we have to check that if Δ_ϵ satisfies (24), then

$$\frac{1}{1-\gamma} \left[2\alpha\Delta_\epsilon + \gamma \frac{\alpha}{1-\gamma\beta} 2\beta\Delta_\epsilon \right] \leq \frac{\epsilon}{2}$$

holds. This is indeed true, which can be easily verified by substitution.

Now, for $\gamma\beta > 1$, by the explicit expression of $\bar{\omega}$ (Lemma 1), we have to check that

$$2\alpha\Delta_\epsilon + \gamma c (\Delta_\epsilon 2\beta)^{\log_\beta(1/\gamma)} \leq \frac{\epsilon(1-\gamma)}{2},$$

holds, if Δ_ϵ satisfies (25). Since $\log_\beta(1/\gamma) < 1$ in this case, the sufficient condition is⁵

$$(\Delta_\epsilon)^{\log_\beta(1/\gamma)} (2\alpha + \gamma c (2\beta)^{\log_\beta(1/\gamma)}) \leq \frac{\epsilon(1-\gamma)}{2}.$$

This last inequality is satisfied with equality, by substituting (25), and that completes the proof of the lemma. \square

7.4 The Cardinality of Δ_ϵ -Supported Kernel Sets

Before proving the mistake bounds, we provide an upper bound on the number of kernels $N_\epsilon = |\Phi_\epsilon|$.

Lemma 6. *For a fixed uniform splitting scheme, with parameters K and λ , and a single initial grid kernel set Φ_0 , we have that*

$$N_\epsilon \leq |\Phi_0| K \left(\frac{\Delta_{max}}{\Delta_\epsilon} \right)^{\log_{1/\lambda}(K)}.$$

Proof. For every $\phi \in \Phi_0$, consider performing $k(\phi)$ splits iteratively, such that at each iteration we obtain new K kernels instead of the original one. It follows that after $k(\phi)$ such splits, the size of a split kernel ϕ' satisfies $\Delta(\phi') \leq \lambda^{k(\phi)} \Delta(\phi)$. In addition, the size of the kernel set that contains all such kernels ϕ' is

$$N = \sum_{\phi \in \Phi_0} K^{k(\phi)}. \quad (36)$$

Thus, for each $\phi \in \Phi_0$, we need to find the minimal $k(\phi)$, such that

$$\lambda^{k(\phi)} \Delta(\phi) \leq \Delta_\epsilon. \quad (37)$$

From (37), it follows that this minimal $k(\phi) = k^*(\phi)$ satisfies

$$\log_{1/\lambda} \left(\frac{\Delta(\phi)}{\Delta_\epsilon} \right) \leq k^*(\phi) < \log_{1/\lambda} \left(\frac{\Delta(\phi)}{\Delta_\epsilon} \right) + 1.$$

Substituting the last inequality in (36) yields

$$\begin{aligned} N_\epsilon &= \sum_{\phi \in \Phi_0} K^{k^*(\phi)} \leq K \sum_{\phi \in \Phi_0} (K)^{\log_{1/\lambda} \left(\frac{\Delta(\phi)}{\Delta_\epsilon} \right)} \\ &= K \sum_{\phi \in \Phi_0} \left(\frac{\Delta(\phi)}{\Delta_\epsilon} \right)^{\log_{1/\lambda}(K)} \leq |\Phi_0| K \left(\frac{\Delta_{max}}{\Delta_\epsilon} \right)^{\log_{1/\lambda}(K)}, \end{aligned}$$

which completes the proof. \square

Remark. We note that Lemma 6 shows an *exponential* dependence of N_ϵ on the state space dimension n since in most cases $\log_{1/\lambda}(K)$ is of order of n .

⁵This is true, of course, for sufficiently small ϵ .

7.5 Proof of Theorem 1

First, we note that the escape event $E_t(x)$ (Definition 10) can be viewed as an *exploration* event. If it occurs at some time $t \geq 0$, the algorithm will encounter (in an execution of length $T_{\epsilon/2}$) a kernel ϕ which is not sampled and/or is not in Φ_ϵ (that is, $\Delta(\phi) > \Delta_\epsilon$). This fact can be interpreted as a “discovery” of new information, since every such occurrence of an “unknown” kernel will lead to an increase of the count of the kernel, and, eventually, to the split of the kernel.

Next lemma shows that the number of times that the escape event can occur is bounded.

Lemma 7. *The number of times that $E_t(x)$ can occur is bounded by $N_\epsilon |\mathbb{A}| (\mathcal{N} + 1) T_{\epsilon/2}$.*

Proof. Note that any kernel $\phi \in \Phi_t(a)$ for any a and t , can be visited no more than \mathcal{N} times – after this number of times, the kernel is split. Now think of the kernel-based representation of the state space as a tree, with kernels as leaves. The internal nodes in such tree represent larger kernels, that were used in previous episodes. Now, the number of such internal nodes is less or equal to the number of leaves, since the splitting coefficient is greater or equal to 2. Using this tree representation, the visit to the “unknown” kernel can be interpreted as a visit to *an internal node of Φ_ϵ* . Since the counter of this kernel is incremented in this visit, by a simple counting argument (a.k.a. the Pigeonhole Principle), the number of times that the algorithm can encounter an internal node of Φ_ϵ is bounded by

$$(\text{number of internal nodes of } \Phi_\epsilon) \cdot \mathcal{N} \cdot |\mathbb{A}| \leq N_\epsilon \mathcal{N} |\mathbb{A}|.$$

Finally, when the algorithm encounters a leaf of Φ_ϵ , then only one such occurrence is sufficient in order to the kernel to become sampled. Again, by a simple counting argument, the number of times this can occur is bounded by

$$(\text{number of leaves of } \Phi_\epsilon) \cdot |\mathbb{A}| = N_\epsilon |\mathbb{A}|.$$

To conclude, the number of times that an “unknown” kernel can be encountered is bounded by $N_\epsilon \mathcal{N} |\mathbb{A}| + N_\epsilon |\mathbb{A}| = N_\epsilon (\mathcal{N} + 1) |\mathbb{A}|$.

At each time t , consider the execution of a (non-stationary) policy \mathcal{A}_t for $T_{\epsilon/2}$ time steps in M . We have two mutually exclusive cases:

- (a) If starting at time t , we execute the policy \mathcal{A}_t for $T_{\epsilon/2}$ time steps, without encountering an “unknown” kernel (that is, a pair (ϕ, a) not in \mathcal{K}_t), there is no occurrence of the escape event $E_t(x)$.
- (b) If starting at time t , we execute the policy \mathcal{A}_t for $T_{\epsilon/2}$ time steps, and encounter at least one unknown kernel at time $t \leq t' \leq t + T_{\epsilon/2}$, the escape event $E_t(x)$ occurs.

We then wish to bound the number of time steps that (b) is the case. In the worst case we will encounter an unknown kernel at the end of the execution episode of length $T_{\epsilon/2}$. In this case, we have that all the succeeding executions for $t < t' \leq t + T_{\epsilon/2}$ will also encounter this unknown kernel. That is, if $E_t(x_t)$ occurs at some time t , also $E_{t'}(x_{t'})$ for $t < t' \leq t + T_{\epsilon/2}$ will occur, in the worst case. Since after $N_\epsilon (\mathcal{N} + 1) |\mathbb{A}|$ visits to unknown kernels, all the kernels will become known, $E_t(x)$ can occur at most $N_\epsilon (\mathcal{N} + 1) |\mathbb{A}| T_{\epsilon/2}$ times. \square

Finally, we prove the main theorem regarding the mistake bound of the AAA algorithm.

Proof of Theorem 1. For each time t , we consider the execution of policy \mathcal{A}_t for $T_{\epsilon/2}$ time-steps in M , with the initial state in each such execution x_t . We then have that

$$\begin{aligned} J_M^{\mathcal{A}_t}(x_t) &\geq \tilde{V}_t(x_t) - \epsilon - \mathbb{I}\{E_t(x_t)\} V_b \\ &\geq V(x_t) - \epsilon - \mathbb{I}\{E_t(s_t)\} V_b, \end{aligned}$$

where the first inequality holds by Lemma 4, and the second inequality holds by Lemma 3. However, by Lemma 7, the number of times the event $E_t(x_t)$ can occur is bounded by $N_\epsilon (2\mathcal{N} + 1) |\mathbb{A}| T_{\epsilon/2}$, implying that

$$\sum_{t=0}^{\infty} \mathbb{I}\{J_M^{\mathcal{A}_t}(x_t) < V(x_t) - \epsilon\} \leq N_\epsilon |\mathbb{A}| (\mathcal{N} + 1) T_{\epsilon/2},$$

which completes the proof of the theorem, using the definition of the $\epsilon/2$ -horizon time, and the fact that $\log_{1/\gamma} C \leq \frac{1}{1-\gamma} \ln C$, for any C . \square

7.6 Proof of Theorem 3

Recall the definition of the posterior number $N_\infty(x_0)$ of actually used kernels in the course of the algorithm (Definition 7). We only need to prove the analogue of Lemma 7 in this case. The rest of the proof is exactly the same as that of Theorem 1.

Thus, we need to bound the number of times that an escape event occurs. Here we consider the trees that represent the kernel sets “at infinity”, namely $\Phi_\infty(a) = \lim_{n \rightarrow \infty} \Phi_t(a)$, $a \in \mathbb{A}$, instead of the Δ_ϵ -supported kernel set Φ_ϵ . As previously, the escape event can occur on *internal* nodes of $\Phi_\infty(a)$ no more than

$$(\text{number of internal nodes of } \Phi_\infty(a)) \cdot \mathcal{N} \leq N_\infty(x_0, a) \mathcal{N}$$

times. The *leaves* of the tree (which are the kernels of $\Phi_\infty(a)$) can be classified into two groups: (a) “Small” leaves, with $\Delta(\phi) \leq \Delta_\epsilon$; and (b) “Large” leaves, with $\Delta(\phi) > \Delta_\epsilon$. On “small” leaf, only one occurrence of the escape event is possible, since the corresponding kernel becomes known (Definition 9) after one sample. On “large” leaves, there can not be more than \mathcal{N} occurrences of the escape event – otherwise these kernels would have been split. Thus, the number of times the escape event can occur on leaves is bounded by

$$(\text{number of leaves of } \Phi_\infty(a)) \mathcal{N} = N_\infty(x_0, a) \mathcal{N}.$$

To summarize, the number of times that the escape event can occur on all nodes, for all actions $a \in \mathbb{A}$, is bounded by $\sum_{a \in \mathbb{A}} 2N_\infty(x_0, a) \mathcal{N}$.

By the same arguments as in proof of Theorem 1, the above bound times the $\epsilon/2$ -horizon time $T_{\epsilon/2}$ is the mistake bound of the algorithm.

8 Incremental Variant of the Algorithm

The ARL algorithm introduced in Section 4 (Algorithm 1) requires very large computational resources, as the upper value function is computed *exactly* each time the model

is changed (see Algorithm 2, equation (13); also, see Section 5). In this section we propose an incremental variant of the ARL, which we call IARL (standing for Incremental ARL). This algorithm only performs back-ups of Value Iteration at the currently visited kernel-action pairs, instead of the exact calculation. The IARL is inspired by a similar algorithm for finite state spaces, presented in Strehl et al. (2006a). We note that intermediate versions (that perform several back-ups at each time step) can be treated similarly, but are not explicitly addressed here.

Below we present the modifications to the original algorithm. First, at each time t , for every action $a \in \mathbb{A}$, and for each kernel $\phi_i^a \in \Phi_t(a)$, we maintain an (online) estimate $\hat{Q}_t(i, a)$ of $\tilde{Q}_t(i, a) \triangleq \tilde{Q}_t(\phi_i^a)$ (see equation (14)). This estimate is initialized to $\hat{Q}_0(i, a) \equiv V_{max}$. At time t , let

$$U_t(i, a) \triangleq \tilde{r}(\phi_i^a) + \gamma \left[\min_{y \in \text{UI}(\phi_i^a)} \hat{V}_t(y) + \bar{\omega}(2\beta\Delta(\phi_i^a)) \right], \quad (38)$$

where

$$\hat{V}_t(x) \triangleq \max_{a \in \mathbb{A}} \left\{ \sum_{i=1}^{|\Phi_t(a)|} \phi_i^a(x) \hat{Q}_t(i, a) \right\}. \quad (39)$$

Also, our incremental variant of the algorithm receives an additional parameter ϵ_Q , which we call the *upper Q -function accuracy parameter*, to be specified in Theorem 4 below. Then, at time t , \hat{Q}_t is updated as follows:

$$\hat{Q}_{t+1}(i, a) := \begin{cases} U_t(i, a), & \text{if } a_t = a, \ x_t \in \text{supp}(\phi_i^a), \text{ and } U_t(i, a) < \hat{Q}_t(i, a) - \epsilon_Q, \\ \hat{Q}_t(i, a), & \text{otherwise.} \end{cases} \quad (40)$$

That is, the update takes place only for the “visited” kernel-action pair, and only if this update changes the estimate by more than ϵ_Q .

Each time a kernel ϕ is split, the upper Q -function estimates of the new kernels ϕ_1, \dots, ϕ_K are initialized to the estimate of the parent kernel ϕ . The policy will be recomputed each time the estimates change, and it is the greedy policy with respect to \hat{Q}_t :

$$\pi_t(x) = \operatorname{argmax}_{a \in \mathbb{A}} \left\{ \sum_{i=1}^{|\Phi_t(a)|} \phi_i^a(x) \hat{Q}_t(i, a) \right\}. \quad (41)$$

To summarize, we have the following modifications to the ARL algorithm:

1. The UVF computation (15) is replaced by the update rule (40).
2. The policy computation (16) is replaced by (41).
3. The policy is recomputed each time the estimates and/or the model change.
4. The estimates of the new kernels after a split are initialized to the estimate of the parent kernel.

Now we have the following results regarding this modified algorithm.

Theorem 4. Assume that the IARL algorithm receives the input parameter Δ_ϵ as in Theorem 1 as well as $\epsilon_Q = \epsilon(1 - \gamma)$. Then, the prior mistake bound is

$$\text{PMC}(2\epsilon) \leq \left(\frac{|\Phi_\epsilon| |\mathbb{A}| (\mathcal{N} + 1)}{1 - \gamma} + \frac{2 |\Phi_\epsilon| |\mathbb{A}| V_b}{\epsilon(1 - \gamma)} \right) \ln \frac{2V_b}{\epsilon}.$$

Theorem 5. Assume that the IARL algorithm receives the input parameter Δ_ϵ as in Theorem 1 as well as $\epsilon_Q = \epsilon(1 - \gamma)$. Then, the posterior mistake bound is

$$\text{PMC}(2\epsilon) \leq \left(\frac{2N_\infty(x_0)\mathcal{N}}{1 - \gamma} + \frac{2N_\infty(x_0)V_b}{\epsilon(1 - \gamma)} \right) \ln \frac{2V_b}{\epsilon}.$$

Observe that the bounds of Theorems 4 and 5 have an additional term compared to these of Theorems 1 and 3. This term compensates for the inaccuracy that is introduced by using an estimated UVF \hat{V} for policy computation, instead of the exact UVF \tilde{V} . Note that the order of magnitude of this new term is approximately $1/\epsilon(1 - \gamma)$ times that of the first term (recall the definition of V_b (1)).

9 Analysis of the IARL Algorithm

Below is the outline of the analysis. To start, we show that the estimated UVF \hat{V}_t indeed upper bounds the optimal value V . Then, we prove an analogue to Lemma 4, which states that under certain conditions on the kernel set, the optimal policy with respect to the estimated UVF is an ϵ -optimal policy. Finally, we provide a bound on the number of times that the escape event can occur.

First, we prove that the estimated upper value function (39) is indeed an upper bound on the optimal value, at each time t .

Lemma 8. $\hat{V}_t(s)$ is an upper bound on the optimal value function. That is, at every time t , we have that

$$\hat{V}_t(x) \geq V(x), \quad \forall x \in \mathbb{X}.$$

Proof. This claim is easily proved by induction on t . First, since $\hat{Q}_0(i, a) \equiv V_{\max}$, the base case is satisfied. Now, assume that $\hat{Q}_t(i, a) \geq Q(x, a)$ for all i such that $x \in \text{supp}(\phi_i^a)$, where Q is the optimal Q -function. For time $t + 1$, if the update of the estimated upper Q -function takes place for (i, a) , we have for all $x \in \text{supp}(\phi_i^a)$, that

$$\hat{Q}_{t+1}(i, a) = \tilde{r}(\phi_i^a) + \gamma \left[\min_{y \in \text{UI}(\phi_i^a)} \hat{V}_t(y) + \bar{\omega}(2\beta\Delta(\phi_i^a)) \right] \geq r(x, a) + \gamma V(f(x, a)) \triangleq Q(x, a),$$

where the inequality follows by the continuity assumption and the induction assumption, similarly to the proof of Lemma 3. Finally, if the update does not take place for (i, a) , it holds that $\hat{Q}_{t+1}(i, a) = \hat{Q}_t(i, a) \geq Q(x, a)$ by the induction assumption. We thus have shown that

$$\hat{Q}_t(i, a) \geq Q(x, a), \quad \forall t, \quad \forall (i, a) : x \in \text{supp}(\phi_i^a).$$

Hence,

$$\hat{V}_t(x) \triangleq \max_{a \in \mathbb{A}} \sum_{i=1}^{|\Phi_t(a)|} \phi_i^a(x) \hat{Q}_t(i, a) \geq \max_{a \in \mathbb{A}} Q(x, a) \triangleq V(x), \quad , \quad \forall t, \forall x \in \mathbb{X}.$$

□

To proceed, we need to obtain an analogue to Lemma 4. To that end, we introduce the following modified definitions of the known kernel-action pairs and the episode.

Definition 12 (Episode). *An episode is a maximal period of time $[t_0, t_1]$, in which:*

(i) *All visited state-action pairs (x_t, a_t) at times $t = t_0, \dots, t_1 - 1$ satisfy that for each kernel $\phi_i^{a_t} \in \Phi_t(a_t)$ with $x_t \in \text{supp}(\phi_i^{a_t})$, it holds that*

$$(a) \quad \Delta(\phi_i^{a_t}) \leq \Delta_\epsilon,$$

(b) *$\phi_i^{a_t}$ was previously sampled (that is, was previously visited).*

(c) *The upper Q -function estimate for $\phi_i^{a_t}$ satisfies:*

$$U_t(i, a_t) \geq \hat{Q}_t(i, a_t) - \epsilon_Q,$$

where U_t is defined in (38).

(ii) *At time $t = t_1$, the algorithm encounters a pair for which the conditions in (i) are not true.*

Definition 13 (Known Pairs). *At any time t , we define the set of known kernel-action pairs:*

$$\mathcal{K}_t \triangleq \left\{ (\phi_i^a, a) \in \Phi_t(a) \times \mathbb{A} : \Delta(\phi_i^a) \leq \Delta_\epsilon, \phi_i^a \text{ was previously sampled, and } \hat{Q}_t(i, a) - U_t(i, a) < \epsilon_Q \right\}.$$

The escape event is defined as previously (Definition 10). We note that the modified definition of the known pairs (and that of the escape event) depends on the size threshold parameter Δ_ϵ and on the upper Q -function accuracy parameter ϵ_Q . We then have the following.

Lemma 9. *Let $\epsilon > 0$ be given. Assume the the incremental variant receives an input Δ_ϵ as in Lemma 4, and an input $\epsilon_Q = \epsilon(1 - \gamma)$. Then,*

$$\hat{V}_t(x) - J_M^{A_t}(x) \leq 2\epsilon + \mathbb{I}\{E_t(x)\}V_b$$

holds for all t and $x \in \mathbb{X}$.

Proof. Similarly to the proof of Lemma 4, at given time t_0 , we consider the execution of the (non-stationary) policy \mathcal{A}_{t_0} for $T_{\epsilon/2}$ time steps in M . As previously, the escape event is expressed by the $\mathbb{I}\{E_{t_0}(x)\}V_b$ term in the bound.

Now, if during this execution we are visiting only known kernel-action pairs, for $T_{\epsilon/2}$ time steps we stay in the same episode, and thus the algorithm's policy remains unchanged and it is the stationary policy π_{t_0} . Assume $t_0 = 0$, write π for π_0 and \hat{V} for \hat{V}_0 , and recall that π is the greedy policy with respect to \hat{V} . Thus,

$$\begin{aligned} \hat{V}(x_0) &= \sum_{\phi_i^{a_0} \in \Phi(a_0)} \phi_i^{a_0}(x_0) \hat{Q}(i, a_0) \\ &\leq \sum_{\phi_i^{a_0} \in \Phi(a_0)} \phi_i^{a_0}(x_0) \left[\epsilon_Q + \tilde{r}(\phi_i^{a_0}) + \gamma \left(\min_{y \in \text{UI}(\phi_i^{a_0})} \hat{V}(y) + \bar{\omega}(2\beta\Delta(\phi_i^{a_0})) \right) \right] \\ &= \epsilon_Q + \sum_{\phi_i^{a_0} \in \Phi(a_0)} \phi_i^{a_0}(x_0) \left[\tilde{r}(\phi_i^{a_0}) + \gamma \left(\min_{y \in \text{UI}(\phi_i^{a_0})} \hat{V}(y) + \bar{\omega}(2\beta\Delta(\phi_i^{a_0})) \right) \right], \end{aligned}$$

where the inequality follows by the modified definition of the actually know kernel-action pairs (Definition 13). Also, by Bellman's equation,

$$J_M^\pi(x_0) = r(x_0, a_0) + \gamma J_M^\pi(x_1).$$

Thus, as in the proof of Lemma 4, we have that

$$\begin{aligned} \widehat{V}(x_0) - J_M^\pi(x_0) &\leq \epsilon_Q + 2\alpha\Delta_\epsilon + \gamma \left(\sum_{\phi \in \Phi(a_0)} \phi(x_0) \left(\min_{y \in \text{UI}(\phi)} \widehat{V}(y) + \bar{\omega}(2\beta\Delta_\epsilon) \right) - J_M^\pi(x_1) \right) \\ &\leq \epsilon_Q + 2\alpha\Delta_\epsilon + \gamma \left(\left(\widehat{V}(x_1) + \bar{\omega}(2\beta\Delta_\epsilon) \right) \left[\sum_{\phi \in \Phi(a_0)} \phi(x_0) \right] - J_M^\pi(x_1) \right) \\ &\leq \epsilon_Q + 2\alpha\Delta_\epsilon + \gamma \bar{\omega}(2\beta\Delta_\epsilon) + \gamma \left(\widehat{V}(x_1) - J_M^\pi(x_1) \right). \end{aligned}$$

Proceeding iteratively, we obtain that

$$\begin{aligned} \widehat{V}(x_0) - J_M^\pi(x_0) &\leq \sum_{t=0}^{T_{\epsilon/2}-1} \gamma^t \left[\epsilon_Q + 2\alpha\Delta_\epsilon + \gamma \bar{\omega}(2\beta\Delta_\epsilon) \right] + \gamma^{T_{\epsilon/2}} V_b \\ &\leq \frac{\epsilon_Q}{1-\gamma} + \frac{1}{1-\gamma} \left[2\alpha\Delta_\epsilon + \gamma \bar{\omega}(2\beta\Delta_\epsilon) \right] + \frac{\epsilon}{2}. \end{aligned}$$

Now the result follows by the precondition of the lemma on Δ_ϵ (see condition 33), and by using $\epsilon_Q = \epsilon(1-\gamma)$. \square

To complete the analysis, it is left to prove a bound on the number of times that the escape event can occur.

Lemma 10. *The number of times that $E_t(x)$ can occur is bounded by*

$$\left(N_\epsilon |\mathbb{A}| (\mathcal{N} + 1) + \frac{2N_\epsilon |\mathbb{A}| V_b}{\epsilon_Q} \right) T_{\epsilon/2}.$$

Proof. The first term in the above bound is exactly the same as that of Lemma 7. The second term addresses the modification of the definition of known kernel-action pairs in the incremental variant (Definition 13). Below we focus on each occurrence of the escape event, in which we have for the visited pair (ϕ_i^a, a) that $\widehat{Q}_t(i, a) - U_t(i, a) > \epsilon_Q$. Now, since when splitting is performed, we initialize the children kernels using the estimate of the parent kernel, and since we cannot decrease the estimate below its minimal possible value V_{\min} , it follows that the number of times that an update $\widehat{Q}_{t+1}(i, a) := U_t(i, a)$ can occur in a situation when $\widehat{Q}_t(i, a) - U_t(i, a) > \epsilon_Q$, is bounded by V_b/ϵ_Q . We wish to bound the number of such updates for all possible kernel-action pairs in the course of learning. Hence, the above bound should be multiplied by $2N_\epsilon |\mathbb{A}| T_{\epsilon/2}$, using similar arguments to those of the proof of Lemma 7. \square

Finally, we prove the prior and the posterior mistake bounds for the incremental variant.

Proof of Theorem 4. For each time t , we consider the execution of policy \mathcal{A}_t for $T_{\epsilon/2}$ time-steps in M , with the initial state in each such execution x_t . We then have that

$$\begin{aligned} J_M^{\mathcal{A}_t}(x_t) &\geq \widehat{V}_t(x_t) - 2\epsilon - \mathbb{I}\{E_t(x_t)\} V_b \\ &\geq V(x_t) - 2\epsilon - \mathbb{I}\{E_t(x_t)\} V_b, \end{aligned}$$

where the first inequality holds by Lemma 9, and the second inequality holds by Lemma 8. However, by Lemma 10, the number of times the event $E_t(x_t)$ can occur is bounded by $N_\epsilon (\mathcal{N} + 1) |\mathbb{A}| T_{\epsilon/2} + \frac{2N_\epsilon |\mathbb{A}| V_b}{\epsilon_Q}$, implying that

$$\sum_{t=0}^{\infty} \mathbb{I}\{J_M^{\mathcal{A}_t}(x_t) < V(x_t) - 2\epsilon\} \leq \left(N_\epsilon |\mathbb{A}| (\mathcal{N} + 1) + \frac{2N_\epsilon |\mathbb{A}| V_b}{\epsilon(1 - \gamma)} \right) T_{\epsilon/2},$$

which completes the proof of the theorem, using the definition of the $\epsilon/2$ -horizon time, and the fact that $\log_{1/\gamma} C \leq \frac{1}{1-\gamma} \ln C$, for any C . \square

Proof of Theorem 5. The proof of the posterior mistake bound of the incremental variant is similar to that of Theorem 3, with the modified definition of known cell-action pairs, as it was done above for the prior bound. \square

10 Conclusion

We presented a learning algorithm which combines a model-based approach and kernel-based value function approximation, in order to solve the online, continuous state space reinforcement learning problem in deterministic domain, under continuity assumption of model parameters. The goal of the analysis was to show feasibility in the sense of sample efficiency. To our best knowledge, this is the first online performance bound result for kernel-based RL algorithm in continuous state space. In addition, an incremental variant was presented and analyzed. This variant is more a practical version of the original algorithm, with reduced computational complexity.

Some ideas for improvement of the proposed algorithm and its analysis follow. First, the explicit dependence of the posterior number of kernels (N_∞ in Definition 7) on problem at hand remains an open and difficult question which requires new tools for its analysis. In addition, more elaborate splitting rules and possible merging schemes should be considered, as in Munos and Moore (2002); Bonarini et al. (2007). Evaluation of the algorithm and possible variants using simulations would be interesting from the practical point of view.

The extension of similar ideas to the stochastic domains seems possible, under a different continuity assumption (namely, under continuity of *transition density* as in Chow and Tsitsiklis (1991)). Preliminary results for the special case of *adaptive aggregation* can be found in Bernstein (2007). A possible future direction here is to formulate an algorithm that will work for both the stochastic and deterministic cases, under a unified continuity assumption.

The model used in this paper assumes a *finite* action space. We note that the proposed algorithm can be applied also to continuous action space by using some fixed (predefined) discretization of the action space. More efficient algorithm may possibly

be obtained by using *adaptive* aggregation of the action space, similar to that of the state space.

Finally, other reward criteria should be considered – average reward (with associated loss bounds), and shortest path problems (total reward). In particular, the shortest path formulation is often associated with such deterministic problems, as navigation in a maze.

Appendix

Below we discuss two important special cases of the general kernel-based framework, where the UVF computation procedure proposed in Section 5 can be carried out in an efficient way.

Constant Kernels – State Aggregation

Consider the important special case where *constant* kernels are used. Namely, for all $a \in \mathbb{A}$ and $\phi_i^a \in \Phi_t(a)$ we have that

$$\phi_i^a(x) = \mathbb{I}\{x \in \text{supp}(\phi_i^a)\},$$

and only one kernel is active for each x . These kernels correspond to B-splines of the degree 0, in the terminology of Unser (1999). Observe that our general framework in this case reduces to (adaptive) grid-based representation of the state space, since each “kernel” aggregates a part of the state space, and there is no overlap in the supports of different kernels in the same action kernel set. A splitting scheme in this case is simply dividing a cell (that is, the support of the corresponding kernel) into disjoint subcells which cover the original cell.

Also, the computational procedure described in the Section 5 reduces to that of the Adaptive Aggregation Algorithm (AAA) (Bernstein and Shimkin, 2008)⁶. Indeed, the UVF $\tilde{V}_t(x)$ is a piecewise constant function and the minimization in (21) reduces to a minimum of a *finite* number of values. In particular, let

$$\mathbb{S}_t(a) = \{s = \text{supp}(\phi) : \phi \in \Phi_t(a)\}$$

be the grid that is used by the algorithm at time t for action a . We denote by \mathbb{S}_t the coarsest grid which is a refinement of all $\mathbb{S}_t(a)$ at time t . That is

$$\mathbb{S}_t \triangleq \bigwedge_{a \in \mathbb{A}} \mathbb{S}_t(a),$$

where the intersection operator is defined as:

$$A \wedge B \triangleq \{s_A \cap s_B : s_A \in A, s_B \in B\} \setminus \{\emptyset\}.$$

Using this notation, we can rewrite equation (21) for the case of constant kernels as follows:

$$\tilde{T}(s, a) = \min_{s' \in \text{UI}_f(s, a)} \max_{a' \in \mathbb{A}} \left\{ \tilde{r}(s', a') + \gamma \left[\tilde{T}(s', a') + \bar{\omega}(s', a') \right] \right\}, \quad (s, a) \in \mathbb{S}_t(a) \times \mathbb{A}. \quad (42)$$

⁶We note however that the proposed algorithm differs from the AAA algorithm as explained in the introduction section.

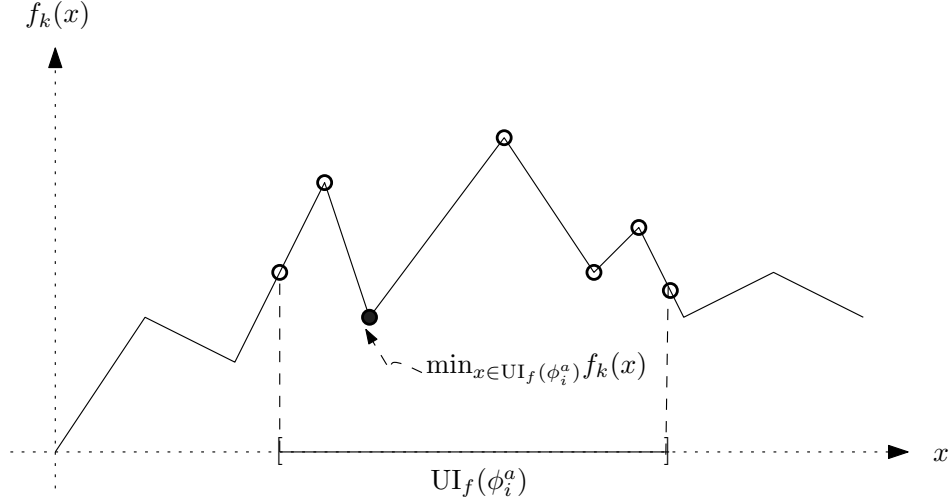


Figure 2: Minimization problem in the case of triangular kernels. Empty circles denote the set $G(i, a)$ of “knot” and boundary points. The minimum of f_k is attained at the point denoted by the filled circle.

Here, $\text{UI}_f : \mathbb{S}_t(a) \times \mathbb{A} \rightarrow \mathbb{S}_t$, $\tilde{r} : \mathbb{S}_t(a) \times \mathbb{A} \rightarrow \mathbb{R}$, and $\bar{\omega} : \mathbb{S}_t(a) \times \mathbb{A} \rightarrow \mathbb{R}$ are corresponding uncertainty parameters (obtained similarly to those of the general kernel-based framework) which are naturally extended to $\mathbb{S}_t \times \mathbb{A}$. Now, observe that since the minimum and maximum in (42) are over finite sets, the related computational procedure can be carried out efficiently.

Triangular Kernels – Piecewise-Linear Approximation

Here we consider the special case where *triangular* kernels are used. An example of such kernel is shown in Figure 1 for one-dimensional case. These kernels correspond to B-splines of the degree 1. The construction of higher-dimensional triangular kernels can be done for example using tensor-product basis functions as in Unser (1999).

Uniform splitting schemes in this case should be considered. One possibility is to use a standard B-spline “pyramids” as discussed in Section 3.2 (see Figure 1). We note that other splitting schemes, which satisfy Definition 3 are also possible.

Regarding the policy computation procedure, observe that in this case the UVF $\tilde{V}_t(x)$ is a piecewise linear function (as it is a finite maximum of piecewise linear functions by its definition). Thus, the minimum in (21) is achieved at the non-differentiable points of its argument (that is, at the “knot” points). This fact may considerably simplify the computation of (21), since the cardinality of the set of such knot points is much smaller than cardinality of UI_f .

For instance, consider the one-dimensional case, recall the general computational procedure (22), and set

$$f_k(x) \triangleq \max_{a' \in \mathbb{A}} \left\{ \sum_{j=1}^{|\Phi_t(a')|} \phi_j^{a'}(x) \left(\tilde{r}(\phi_j^{a'}) + \gamma \left[\tilde{T}^{(k-1)}(j, a') + \bar{\omega}(j, a') \right] \right) \right\}.$$

Note that $f_k(x)$ is a piecewise-linear function since it is a finite maximum of linear

combination of triangular kernels. Now, the update required in (22) is

$$\tilde{T}^{(k)}(i, a) = \min_{x \in \text{UI}_f(\phi_i^a)} f_k(x),$$

which is a minimum of the piecewise-linear function $f_k(x)$ over the *convex* set $\text{UI}_f(\phi_i^a)$. Let

$$G(i, a) = \{x \in \text{UI}_f(\phi_i^a) : f_k(x) \text{ is non-differentiable at } x\} \cup \partial \text{UI}_f(\phi_i^a),$$

where $\partial \text{UI}_f(\phi_i^a)$ is the boundary of $\text{UI}_f(\phi_i^a)$. However, $|G(i, a)|$ is *finite* since $f_k(x)$ is piecewise-linear. Thus, the update

$$\tilde{T}^{(k)}(i, a) = \min_{x \in \text{UI}_f(\phi_i^a)} f_k(x) = \min_{x \in G(i, a)} f_k(x)$$

can be carried out efficiently, as illustrated by Figure 2.

References

- J. S. Albus. A new approach to manipulator control: the cerebellar model articulation controller (CMAC). *Journal of Dynamic Systems, Measurement and Control*, 97: 220–227, 1975.
- A. Antos, C. Szepesvári, and R. Munos. Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, 71(1):89–129, 2008.
- P. Auer and R. Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. In *Proceedings of Neural Information Processing Systems Conference (NIPS)*, 2006.
- A. Bernstein. Adaptive state aggregation for reinforcement learning. Master’s thesis, Technion – Israel Institute of Technology, 2007. URL: http://tx.technion.ac.il/~andreyb/MSc_Thesis_final.pdf.
- A. Bernstein and N. Shimkin. Adaptive aggregation for reinforcement learning with efficient exploration: Deterministic domains. In *Proceedings of the 21st Annual Conference on Learning Theory (COLT 2008)*, 2008.
- D. P. Bertsekas. *Dynamic Programming and Optimal Control, vol. 2*. Athena Scientific, Belmont, MA, third edition, 2007.
- A. Bonarini, A. Lazaric, and M. Restelli. LEAP: an adaptive multi-resolution reinforcement learning algorithm. *Journal of Machine Learning Research*, 2007. conditionally accepted.
- C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11: 1–94, 1999.
- R. I. Brafman and M. Tennenholtz. R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3: 213–231, 2002.

- H. Chapman. Global confidence bound algorithms for the exploration-exploitation tradeoff in reinforcement learning. Master’s thesis, Technion – Israel Institute of Technology, 2007.
- C.-S. Chow and J.N. Tsitsiklis. An optimal one-way multigrid algorithm for discrete-time stochastic control. *IEEE Transactions on Automatic Control*, 36(8):898–914, 1991.
- K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12:219–245, 2000.
- Y. Engel, S. Mannor, and R. Meir. Bayes meets Bellman: The Gaussian process approach to temporal difference learning. In *Proceedings of the 20th International Conference on Machine Learning (ICML 2003)*, 2003.
- N. Jong and P. Stone. Kernel-based models for reinforcement learning in continuous state spaces. In *23th International Conference on Machine Learning (ICML 2006), Workshop on Kernel Machines and Reinforcement Learning*, 2006.
- S. M. Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London, UK, 2003.
- M. Kearns and S. P. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49:209–232, 2002.
- M. Loth, M. Davy, R. Coulom, and P. Preux. Equi-gradient temporal difference learning. In *23th International Conference on Machine Learning (ICML 2006), Workshop on Kernel Machines and Reinforcement Learning*, 2006.
- A. W. Moore and C. G. Atkeson. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning*, 21:199–233, 1995.
- R. Munos and A. W. Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49:291–323, 2002.
- D. Ormoneit and S. Sen. Kernel-based reinforcement learning. *Machine Learning*, 49:161–178, 2002.
- W. B. Powell. *Approximate Dynamic Programming for Operations Research: Solving the curses of dimensionality*. John Wiley & Sons, Inc., New York, NY, USA, 2007.
- M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994.
- J. Reisinger, P. Stone, and R. Miikkulainen. Online kernel selection for Bayesian reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, 2008.
- S. P. Singh, T. Jaakkola, and M. I. Jordan. Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing Systems (NIPS) 7*, pages 361–368, 1995.

- A. L. Strehl and M. L. Littman. A theoretical analysis of model-based interval estimation. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 857–864, 2005.
- A. L. Strehl, L. Li, and M. L. Littman. Incremental model-based learners with formal learning-time guarantees. In *Proceedings of the 22nd International Conference on Uncertainty in Artificial Intelligence*, pages 485–493, 2006a.
- A. L. Strehl, E. Wiewiora, J. Langford, and M. L. Littman. PAC model-free reinforcement learning. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 881–888, 2006b.
- R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8 (NIPS)*, pages 1038–1044, 1996.
- A. Tewari and P. L. Bartlett. Optimistic linear programming gives logarithmic regret for irreducible MDPs. In *Proceedings of Neural Information Processing Systems Conference (NIPS)*, 2007.
- M. Unser. Splines: A perfect fit for signal and image processing. *IEEE Signal Processing Magazine*, 16:22–38, 1999.
- W. Whitt. Approximations of dynamic programs, I. *Mathematics of Operations Research*, 3(3):231–243, 1978.