

---

# Adaptive Aggregation for Reinforcement Learning with Efficient Exploration: Deterministic Domains

---

**Andrey Bernstein\***

Department of Electrical Engineering  
Technion – Israel Institute of Technology  
Haifa, Israel  
andreyb@tx.technion.ac.il

**Nahum Shimkin**

Department of Electrical Engineering  
Technion – Israel Institute of Technology  
Haifa, Israel  
shimkin@ee.technion.ac.il

## Abstract

We propose a model-based learning algorithm, the Adaptive Aggregation Algorithm (AAA), that aims to solve the online, continuous state space reinforcement learning problem in a deterministic domain. The proposed algorithm uses an adaptive state aggregation approach, going from coarse to fine grids over 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 making tasks under uncertainty ([4, 17]). Solid theory and a variety of algorithms enable the efficient computation of 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 [4] and [6] for a thorough discussion. Also, see such recent works as [1, 11, 14].

One natural approximation approach is *state aggregation*, in which the state space is discretized into a (relatively small) finite collection of *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 minimize the problem complexity. This question was addressed by many works, such as [21, 9], 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 model of the MDP 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  $E^3$  [13], R-MAX [7], MBIE [18], GCB [8], UCRL [2] and OLP [20] 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” problems that rely on state aggregation, are heuristic in nature, without formal guarantees. These include such works on adaptive aggregation as [15], [16] and [5]. An exception is the algorithm proposed by Diuk et al. [10], which uses R-MAX as the basis. However, this algorithm requires a specific structure of the problem; otherwise, its total mistake bound is polynomial in the size of the state space, which can be very large or infinite.

In this paper we focus on the online reinforcement learning problem in MDPs with very large or infinite state space, finite action space, discounted return criterion, and with *deterministic* dynamics and rewards. 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 state aggregation* approach, going from coarse to fine grids over 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<sup>1</sup>. Certain continuity assumptions on the ba-

---

\*We would like to thank the reviewers for their useful comments on this paper.

---

<sup>1</sup>Our uncertainty intervals are the analogue of the *confidence intervals* used in the stochastic case [18]. However, the origin of

basic model parameters will be imposed. Such assumptions are essential for generalization in continuous state space, especially when using the state aggregation approach.

The principle that governs our basic scheme is simply *to split frequently visited cells*. The idea behind this principle is as follows. As time progresses, we will visit cells that are “close” to the optimal trajectory; on the optimal trajectory, we need high resolution. Perhaps surprisingly, this principle is not sufficient to obtain theoretical results. Consequently, we will propose an improved variant of the basic algorithm, for which learning rates in terms of total mistake bound (see below) will be established. In this variant, in addition to splitting the visited cells, we also split cells that the algorithm “could have” visited (according to the uncertainty in the model of the MDP that was learned so far).

We will use the *total mistake bound* as a performance metric for our algorithms. This metric counts the total number of time-steps in which the algorithm’s implemented policy is strictly suboptimal from the current state. This metric has been used in a number of recent works on on-line learning in discounted MDP problems<sup>2</sup> [12, 18, 19]. 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 *uniformly dense* grid. In this case, our mistake bound is thus polynomial in the number of cells in this grid. The second type ensures that the mistake bound is polynomial in the number of cells in the *actually used* grid.

In our analysis, we need to distinguish between two cases: The “contractive” case, characterized by  $\gamma\beta < 1$ , where  $\gamma$  is the MDP discount factor and  $\beta$  is a (Lipschitz) continuity parameter of the transition function (cf. equation (6b)); and the “expansive” case, where  $\gamma\beta > 1$ . Due to space constraints, we treat here in detail the former, while the results for the latter are presented without proofs, which can be found in [3].

The paper is structured as follows. In Section 2 we present the model and the notation. In Section 3 we introduce some further definitions and assumptions. In Section 4 we propose a basic version of our AAA algorithm, while Section 5 presents an improved variant of the algorithm that is required for its convergence. In Section 6 polynomial bounds on the total mistake count of this improved algorithm are presented for the “contractive” case, while Section 7 proves these bounds. In Section 8 we provide results for the “expansive” case, without proof. Finally, conclusions and future work are presented in Section 9.

## 2 Model and Performance Metrics

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  $r(x, a) \in [r_{min}, r_{max}]$  is the immediate reward function

uncertainty in our model is the (deterministic) aggregation error, rather than stochastic sampling error.

<sup>2</sup>These works refer to this metric as the *sample complexity of exploration*.

which specifies the reward of performing action  $a \in \mathbb{A}$  in state  $x \in \mathbb{X}$ .

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  $\mathbb{R}^n$ . That is, 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 an 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  $\pi$  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  $\pi$ , 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  $\pi$  is *optimal* if  $J^\pi(x) = V(x)$  holds for all  $x \in \mathbb{X}$ . For any  $\epsilon > 0$ , a policy  $\pi$  is  $\epsilon$ -*optimal* if  $J^\pi(x) \geq V(x) - \epsilon$  holds for all  $x \in \mathbb{X}$ .

It is well known ([17]) that the optimal value function satisfies Bellman’s equation

$$V(x) = \max_{a \in \mathbb{A}} \{r(x, a) + \gamma V(f(x, a))\}, \quad x \in \mathbb{X}, \quad (1)$$

and any stationary deterministic policy  $\pi^*$  which satisfies  $\pi^*(x) \in \operatorname{argmax}_{a \in \mathbb{A}} \{r(x, a) + \gamma V(f(x, a))\}$ ,  $x \in \mathbb{X}$ , 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. Also, we let  $V_{max} \triangleq \frac{r_{max}}{1-\gamma}$ . Note that  $V_{max}$  is the maximal possible discounted return of any policy.

Our main performance metric will be *mistake bound* (or *policy-mistake bound*), introduced for RL in [12]. It counts the number of time steps  $t$  in which the algorithm executes a not  $\epsilon$ -optimal policy from the current state,  $x_t$ . Specifically, let  $\pi_t$  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  $r_k = r(x_k, \pi_k(h_k))$  and  $x_{k+1} = f(x_k, \pi_k(h_k))$ . Now, the policy-mistake count is defined as

$$\text{PM}(\epsilon) \triangleq \sum_{t=0}^{\infty} \mathbb{I} \{ J^{\mathcal{A}_t}(x_t) < V(x_t) - \epsilon \}. \quad (2)$$

For deterministic domains with finite state-space, we have the following near-optimality criterion.

### The Policy-Mistake Bound Criterion

A learning algorithm is PAC (Probably Approximately Correct) if there exists a polynomial

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

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

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

A possible alternative to the policy-mistake count is the action-mistake count, defined as follows:

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

This criterion counts the number of sub-optimal actions, that is, the number of times that an algorithm executed an action whose action-value is  $\epsilon$ -inferior to the optimal value. It is easily verified (see Corollary 1 in [3]) that policy-mistake count is a stronger criterion, in the sense that  $\text{AM}(\epsilon) \leq \text{PM}(\epsilon)$ . Hence we focus here on the former.

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 number of cells in sufficiently dense grid over the state space.

## 3 Preliminaries

### a. Grid-Cell Notation

A grid  $\mathbb{S}$  over the state space  $\mathbb{X}$  is a partition of  $\mathbb{X}$  into disjoint elements that covers the whole of  $\mathbb{X}$ . We call any  $s \in \mathbb{S}$  a cell. We say that a grid  $\mathbb{S}_2$  is a refinement of a grid  $\mathbb{S}_1$ , if for every cell  $s \in \mathbb{S}_2$  there exists  $s' \in \mathbb{S}_1$ , such that  $s \subseteq s'$ . We denote this relation by  $\mathbb{S}_2 \preceq \mathbb{S}_1$ . For any sets  $A$  and  $B$  of cells, we define the *intersection operator* between these two sets as

$$A \wedge B \triangleq \{s_A \cap s_B : s_A \in A, s_B \in B\} \setminus \{\emptyset\}. \quad (4)$$

For a given cell  $s \in \mathbb{S}$ , let  $\Delta(s) \triangleq \sup_{x, x' \in s} d(x, x')$  denote the cell size (or *diameter*) in the given metric  $d$ . For two given cells  $s, s' \in \mathbb{S}$ , we define the *biased distance* between these cells as

$$d_b(s, s') \triangleq \begin{cases} \inf_{x \in s, x' \in s'} d(x, x'), & s \neq s', \\ -\Delta(s), & s = s'. \end{cases} \quad (5)$$

This definition will be justified in Section 5 (see Definition 3 and the remark after this Definition). Of course  $d_b(s, s')$  is not a distance in a regular sense, since it can be negative. Also, for given state  $x \in \mathbb{X}$ , let  $s_x \in \mathbb{S}$  be the cell that includes  $x$  ( $x \in s_x$ ). Then, given a cell  $s \in \mathbb{S}$ , we define the biased state-to-cell distance  $d_b(s, x) \triangleq d_b(s, s_x)$ .

### b. Feasible Splitting Schemes and Grids

Given a source grid  $\mathbb{S}_1$  and a candidate  $s \in \mathbb{S}_1$  to split, a splitting scheme tells us how to split  $s$  into  $c_{split}$  cells  $s_1, \dots, s_{c_{split}}$ , with  $s_i \cap s_j = \emptyset$ ,  $\cup_i s_i = s$ , to form a refined (target) grid  $\mathbb{S}_2 \preceq \mathbb{S}_1$ . We are interested in splitting schemes that decrease the size  $\Delta(s)$  of a cell  $s$ . More formally, we require the following condition on the splitting scheme.

**Definition 1 (Feasible Splitting Scheme)** A splitting scheme with splitting coefficient  $c_{split}$  is feasible if there exists  $0 < \lambda < 1$  (independent of  $s$ ) such that  $\Delta(s_i) \leq \lambda \Delta(s)$  for  $i = 1, \dots, c_{split}$ .

Now, given a fixed feasible splitting scheme and an initial grid  $\mathbb{S}_0$  over the state space  $\mathbb{X}$ , we define the set of *feasible grids* as the set of all grids that can be obtained by using this given scheme starting from  $\mathbb{S}_0$ .

### c. Continuity Assumption

The following continuity assumption will be imposed on 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  $a \in \mathbb{A}$  it holds that

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

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

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 [21, 9]. 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.

In this paper we will treat in detail the case  $\gamma\beta < 1$  (where  $\gamma$  is the discount factor), in which there is some ‘‘contraction’’ effect in the system dynamics. Results for the complementary case of  $\gamma\beta > 1$  are presented without proof in Section 8.

We assume that both  $\alpha$  and  $\beta$  are known for the purpose of learning.

## 4 The Basic AAA Algorithm

In this section we present the basic variant of the AAA algorithm, which is directly based on the principle of splitting frequently visited cells. As it turns out, this algorithm may fail in some cases, and therefore no theoretical guarantees will be presented. Instead, we will provide an example, showing the source of the problem. This will provide the motivation for the improved scheme in the next section.

In the following subsections we present the different parts of this algorithm in detail. An outline of the complete algorithm is presented as Algorithm 1.

---

**Algorithm 1** Basic Adaptive Aggregation Algorithm (outline)

---

**Input parameters:**

Maximal reward  $r_{max}$ ,  
Lipschitz continuity parameters  $\alpha$  and  $\beta$ ,  
Count threshold  $\mathcal{N}$ ,  
Cell size threshold  $\Delta_\epsilon$ .

**Initialization:**

1. Initialize the grid to some initial grid  $\mathbb{S}_0(a) = \mathbb{S}_0$  for all  $a \in \mathbb{A}$ , and the cell count  $N(s, a) = 0$ , for all  $a \in \mathbb{A}, s \in \mathbb{S}_0$ ;
2. For all  $s \in \mathbb{S}_0(a)$  and  $a \in \mathbb{A}$ , initialize the reward upper bound and the transition uncertainty set:

$$\tilde{r}(s, a) = r_{max}, \quad \text{CI}_f(s, a) = \mathbb{S}_0(a).$$

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

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

**a. Action-Grids and Common Grid**

In our algorithm, we will use a separate grid for every action. This will allow to use a different resolution for each action. We denote by  $\mathbb{S}_t(a)$  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 in (4). We call this grid a *common grid* (at time  $t$ ). This grid will be used to compute the value function, while the action-grids are used for empirical model estimation.

**b. Empirical Model**

We use a *single sample* to estimate empirically the reward and transition. Specifically, suppose that we choose action  $a$  in cell  $s$ . We thus obtain the sample  $(x, a, r = r(x, a), x' = f(x, a))$ , with  $x \in s$  and  $x' \in s'$ . We define the empirical model based on this single sample:

$$\hat{r}(s, a) = r, \quad (7)$$

$$\hat{f}(s, a) = s'. \quad (8)$$

Once the sample from  $(s, a)$  is obtained, the model remains unchanged for this pair (until the cell is split).

**c. Uncertainty Intervals and Upper Value Function**

In the AAA algorithm we will use an uncertainty intervals exploration technique as it applies to deterministic systems due to aggregation. Below we present the definition of the uncertainty intervals in case of continuous state space, and how we use them in the algorithm.

At any time  $t$ , and for every  $a \in \mathbb{A}$  and  $s \in \mathbb{S}_t(a)$ , we define the reward uncertainty interval around the empirical reward (7) as<sup>3</sup>:

$$\begin{aligned} \text{CI}_r(s, a) &\triangleq [\underline{r}(s, a), \tilde{r}(s, a)] \\ &= [\hat{r}(s, a) - \alpha\Delta(s), \hat{r}(s, a) + \alpha\Delta(s)] \end{aligned}$$

if the pair  $(s, a)$  was sampled till time  $t$ ; otherwise, the reward uncertainty interval for this pair is inherited from the parent cell. By the continuity Assumption 2, this uncertainty interval satisfies that  $r(x, a) \in \text{CI}_r(s, a)$ ,  $\forall x \in s$ . Also, the transition uncertainty set is defined as:

$$\text{CI}_f(s, a) \triangleq \left\{ s' \in \mathbb{S}_t : d_b(s', \hat{f}(s, a)) \leq \beta\Delta(s) \right\},$$

where  $d_b$  is the biased distance defined in (5). (If the pair  $(s, a)$  was not sampled till time  $t$ , the uncertainty set is inherited from the parent cell as in the reward case). Again, by the continuity assumption, this uncertainty set satisfies:  $f(x, a) \in \text{CI}_f(s, a)$ ,  $\forall x \in s$ . Using this notation, we define the following dynamic programming operator.

**Definition 2** The upper DP operator at time  $t$  for any given function  $g : \mathbb{S}_t \rightarrow \mathbb{R}$  is

$$\mathcal{T}_1 g(s) = \max_{a \in \mathbb{A}} \left\{ \tilde{r}(s, a) + \gamma \max_{s' \in \text{CI}_f(s, a)} g(s') \right\}.$$

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

$$\tilde{V}_t(s) = \mathcal{T}_1 \tilde{V}_t(s), \quad s \in \mathbb{S}_t.$$

It can be shown that this equation has a unique solution, which can be found using Value Iteration or linear programming. Moreover, we can show that this solution is indeed an upper bound on the optimal value function  $V$ . In addition, on dense enough grid this solution is also very close to  $V$ . We do not provide here proofs for these claims. However, these claims easily follow from our analysis of the improved algorithm in Section 7.

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

$$\pi_t(s) = \operatorname{argmax}_{a \in \mathbb{A}} \left\{ \tilde{r}(s, a) + \gamma \max_{s' \in \text{CI}_f(s, a)} \tilde{V}_t(s') \right\}, \quad s \in \mathbb{S}_t.$$

This policy is recalculated only when a cell-action pair is visited for the first time, or some cell is split.

We summarize the UVF and policy computation algorithm in Algorithm 2.

**d. Policy Execution**

As we have seen, the decision rule  $\pi_t$  that is used at each time  $t$ , is determined by the UVF, as presented in Algorithm 2 (equation (10)). In addition, in each execution of the decision rule, a new sample is obtained, and the empirical model and the uncertainty intervals of the corresponding cell-action pairs are updated. This process is summarized in Algorithm 3.

---

<sup>3</sup>We drop the time index from most of our notation for ease of exposition.

---

**Algorithm 2** Policy Computation

---

**If the model has been changed** (that is, some cell-action pair has been visited for the first time, or some cell has been split):

1. Compute the UVF over  $\mathbb{S}_t = \bigwedge_{a \in \mathbb{A}} \mathbb{S}_t(a)$  by solving

$$\tilde{V}_t(s) = \mathcal{T}_1 \tilde{V}_t(s), s \in \mathbb{S}_t, \quad (9)$$

where  $\mathcal{T}_1$  is defined in Definition 2.

2. Compute the corresponding optimal policy

$$\pi_t(s) = \operatorname{argmax}_{a \in \mathbb{A}} \left\{ \hat{r}(s, a) + \gamma \max_{s' \in \text{CI}_f(s, a)} \tilde{V}_t(s') \right\}. \quad (10)$$

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  $\pi_t = \pi_{t-1}$ .

---

### e. Splitting Method

Assume that a fixed feasible splitting scheme is used throughout (cf. Definition 1). Define a count threshold  $\mathcal{N}$ . We will split a cell if the number of visits to it exceeds  $\mathcal{N}$ . In addition to this splitting criterion, we also employ a “stop-splitting” rule, based on the size of the cell. Let  $\Delta_\epsilon$  be a (small) *cell size threshold* parameter. Then, if a cell  $s$  satisfies  $\Delta(s) \leq \Delta_\epsilon$ , it will not be split anymore. Since the number of times that the algorithm encounters a pair with  $\Delta(s) > \Delta_\epsilon$  can be bounded, it follows that the number of different (stationary) policies that the algorithm 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 feasible splitting scheme, we denote by  $\mathbb{S}_\epsilon$  the coarsest feasible grid with  $\Delta(s) \leq \Delta_\epsilon$  for all  $s \in \mathbb{S}_\epsilon$ . We call this grid  *$\epsilon$ -optimal grid*. The number of cells in  $\mathbb{S}_\epsilon$  can be bounded as follows (see Lemma 6 in Section 7):

$$N_\epsilon \triangleq |\mathbb{S}_\epsilon| \leq |\mathbb{S}_0| c_{split} \left( \frac{\Delta_{max}}{\Delta_\epsilon} \right)^{\log_{1/\lambda}(c_{split})}, \quad (13)$$

where  $\mathbb{S}_0$  is the initial grid,  $\lambda$  and  $c_{split}$  are the parameters of the splitting scheme (Definition 1), and  $\Delta_{max}$  is the diameter of the state space (see Assumption 1).

We summarize the splitting process in Algorithm 4. Recall that the complete AAA algorithm is outlined in Algorithm 1.

### f. Why the Basic AAA Scheme Might Fail

To realize the problem, consider some cell  $s$ . The value of the function  $\tilde{V}$  at that cell is computed based on the optimistic next-step cell  $s_1$ :

$$s_1 \triangleq \operatorname{argmax}_{s' \in \text{CI}_f(s, a)} \tilde{V}_t(s') \quad (14)$$

(cf. equation (9) and Definition 2). However, it may happen that the actual process never visits  $s_1$ , but rather some other cell in  $\text{CI}_f(s, a)$ . This may happen irrespectively of

---

**Algorithm 3** Policy Execution

---

- (i) Execute the action  $a_t = \pi_t(s_t)$ , with  $s_t \in \mathbb{S}_t$  being the current common grid cell. For the visited cell-action pair  $(s_t, a_t) = (s_t, a)$ , let  $s \in \mathbb{S}_t(a)$  be the cell in the action-grid that contains  $s_t$ .
  - (ii) Update the counter:  $N(s, a) := N(s, a) + 1$ .
  - (iii) If  $(s, a)$  is visited *for the first time*, compute the model of this pair. Namely,
    - (a) Compute the empirical reward and transition according to equations (7) and (8).
    - (b) Compute the upper reward value
$$\tilde{r}(s, a) := \hat{r}(s, a) + \alpha \Delta(s), \quad (11)$$
    - (c) Compute the transition uncertainty set
$$\text{CI}_f(s, a) := \left\{ s' \in \mathbb{S}_t : d_b(s', \hat{f}(s, a)) \leq \beta \Delta(s) \right\}. \quad (12)$$
    - (d) Save the basic sample  $(x, a, x')$  obtained for this  $(s, a)$ , with  $x = x_t$  and  $x' = x_{t+1}$ .
- 

---

**Algorithm 4** Splitting Algorithm

---

1. Initialize  $\mathbb{S}_{t+1}(a) = \mathbb{S}_t(a)$ , for all  $a \in \mathbb{A}$ .
  2. For each cell-action pair  $(s, a)$ , with  $s \in \mathbb{S}_t(a)$ , which satisfy  $N(s, a) \geq \mathcal{N}$  and  $\Delta(s) > \Delta_\epsilon$ , perform the following:
    - (a) Split this cell-action pair according to the given (feasible) splitting scheme. Let  $s_1, \dots, s_{c_{split}} \in \mathbb{S}_{t+1}(a)$  be the resulting sub-cells after this split. Let  $s_k$  be the cell that contains the sample of the parent cell  $s$ .
    - (b) Initialize the reward upper bounds of the new cells:
$$\tilde{r}(s_j, a) = \tilde{r}(s, a), \quad \forall j \neq k,$$
$$\tilde{r}(s_k, a) = \hat{r}(s, a) + \alpha \Delta(s_k),$$
    - (c) Initialize the transition uncertainty sets of the new cells:
$$\text{CI}_f(s_j, a) = \text{CI}_f(s, a), \quad \forall j \neq k,$$
$$\text{CI}_f(s_k, a) = \left\{ s' \in \mathbb{S}_t : d_b(s', \hat{f}(s, a)) \leq \beta \Delta(s_k) \right\},$$
    - (d) Update the counts of the new cells as follows:
$$N(s_j, a) = 0, \quad \forall j \neq k,$$
$$N(s_k, a) = 1.$$
- 

how small  $s$  is, or how many times it is visited. This might be the case, for example, if some points (states) in  $s$  map under  $f(x, a)$  to the border between  $s_1$  and some adjacent cell  $s_2$ , and all visits to  $s$  are to that part that maps to  $s_2$ . In that case, cell  $s_1$  which is not visited will remain large, hence

with potentially large error in its empirical estimates. This may lead to a large error in the estimated value function at  $s$ , and consequently to an error in the computed policy. We propose a solution to this problem in the next section.

## 5 The AAA Algorithm

To overcome the potential pitfalls of the basic AAA algorithm above, we need to modify the definition of the upper value function so that it will more closely approximate the optimal one. Two modifications will be introduced. First, we propose splitting of the optimistic next-step cells (recall the cell  $s_1$  defined in (14)), in addition to actually visited ones. Those cells, which we call “virtually visited” cells, will be defined formally in Definition 5 below. However, splitting those cells is not enough to fix the problem, since it may happen that no actual samples are obtained for the created cells. Thus, we introduce a *smoothing* operator in the DP equations. This operator, which is specified in Definition 3 below, allows to improve the accuracy of the upper value function in (small) cells even if they are not actually visited (hence not actually sampled), based on the values of their geometric neighbors.

In what follows we will focus on the case  $\gamma\beta < 1$ . As can be seen in the proof of Lemma 1 (Section 7), in this case we have some sort of a contraction effect. Thus, the results are technically much simpler than for  $\gamma\beta > 1$ . The latter case will be briefly discussed in Section 8, and is treated in detail in [3].

**Definition 3** *Let the continuity function of the optimal value be defined as*

$$\omega(\theta) \triangleq \frac{\alpha}{1 - \gamma\beta} \theta, \quad \theta \geq 0.$$

*The smoothing operator at time  $t$  for any given function  $g : \mathbb{S}_t \rightarrow \mathbb{R}$  is*

$$\mathcal{T}_2 g(s) \triangleq \min_{s' \in \mathbb{S}_t} \{g(s') + \omega(\Delta(s) + d_b(s, s'))\}.$$

**Remark.** Note that by definition of the biased distance  $d_b$  in (5), the above minimized set also includes  $g(s)$ , since for  $s' = s$ ,

$$g(s) + \omega(\Delta(s) + d_b(s, s)) = g(s) + \omega(0) = g(s).$$

Therefore,  $\mathcal{T}_2 g(s) \leq g(s)$  for all  $s \in \mathbb{S}_t$ .

It is shown in Lemma 1 in Section 7, that the continuity function  $\omega$  is in fact a bound on the modulus of continuity of the optimal value function  $V$ . The definition of the smoothing operator is then formally justified in Lemma 3, which states that if  $g$  is an upper value function, that is  $g(s) \geq V(x)$  for all  $s$  and  $x \in s$ , then so is  $\mathcal{T}_2 g$ . Thus, the smoothing operator  $\mathcal{T}_2$  tightens the upper value function  $g$  based on the values in adjacent cells.

Now, using the smoothing operator, we modify the definition of the upper DP operator (Definition 2).

**Definition 4** *The smoothed upper DP operator at time  $t$  is defined by  $\tilde{\mathcal{T}} \triangleq \mathcal{T}_1 \mathcal{T}_2$ . That is, for given function  $g : \mathbb{S}_t \rightarrow \mathbb{R}$ ,*

$$\tilde{\mathcal{T}} g(s) = \max_{a \in \mathbb{A}} \left\{ \tilde{r}(s, a) + \gamma \max_{s' \in \text{Cl}_f(s, a)} \mathcal{T}_2 g(s') \right\}.$$

This new operator smooths  $g(s)$  before applying to it the DP operation. As before, we define the UVF as the solution of the fixed point equation:

$$\tilde{V}_t(s) = \tilde{\mathcal{T}} \tilde{V}_t(s), \quad s \in \mathbb{S}_t.$$

Our second modification involves splitting of “virtually visited cells”. We next define the required notation.

**Definition 5 (Virtually Visited Cells)** *At any period  $[t_0, t_1]$  of the algorithm’s execution:*

1. *Let  $\{s_t\}_{t=t_0}^{t_1}$  be the **actually visited cells**  $s_t \in \mathbb{S}_t$  that are visited during this period, and  $\{a_t\}_{t=t_0}^{t_1}$  be the corresponding actions, with  $a_t = \pi_t(s_t)$ .*
2. *Let  $\{(s'_t, a_t)\}_{t=t_0}^{t_1}$  be the **actually visited cell-action pairs** during this period, with  $s'_t \in \mathbb{S}_t(a_t)$ , such that  $s_t \subseteq s'_t$ .*
3. *Denote the **virtually visited cells** during this period by  $\{s_t^*\}_{t=t_0+1}^{t_1+1}$ , where  $s_t^* \in \mathbb{S}_t$  and is the argument of the maximization*

$$s_{t+1}^* \triangleq \operatorname{argmax}_{s' \in \text{Cl}_f(s'_t, a_t)} \mathcal{T}_2 \tilde{V}_t(s').$$

4. *For each virtually visited common grid cell  $s_t^* \in \mathbb{S}_t$ , let  $\{s_a\}_{a \in \mathbb{A}}$  be the action-grid cells ( $s_a \in \mathbb{S}_t(a)$ ) which contain  $s_t^*$ . We define the **virtually visited cell-action pair** as the pair  $(\tilde{s}_t, \tilde{a}_t) = (s_a, a)$  with the cell  $s_a$  having the smallest diameter among those action-grid cells.*

In the course of the algorithm, both actually and virtually visited cell-action pairs will be split (using a count threshold as before). We note that the splitting of virtually visited cells is needed in the common grid  $\mathbb{S}_t$ , to avoid the problem presented in Section 4. This splitting can be done directly in  $\mathbb{S}_t$ . However, we have chosen to keep the relation  $\mathbb{S}_t = \bigwedge_{a \in \mathbb{A}} \mathbb{S}_t(a)$ . Thus, we will split the *smallest* cell  $\tilde{s} \in \mathbb{S}_t(\tilde{a})$  which contains the virtually visited cell  $s^* \in \mathbb{S}_t$  that is candidate for splitting. In this way, the split is inherited by  $\mathbb{S}_t$ .

To summarize, the following modifications are introduced in the AAA algorithm: In Algorithm 2, equation (9) is replaced by

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

where  $\tilde{\mathcal{T}}$  is defined in Definition 4. Also, equation (10) is replaced by

$$\pi_t(s) = \operatorname{argmax}_{a \in \mathbb{A}} \left\{ \tilde{r}(s, a) + \gamma \max_{s' \in \text{Cl}_f(s, a)} \mathcal{T}_2 \tilde{V}_t(s') \right\}. \quad (16)$$

Finally, in Algorithm 3, step (ii), we update the counters of both the actually and virtually visited cell-action pairs:

$$N(s'_t, a_t) := N(s'_t, a_t) + 1, \quad N(\tilde{s}_t, \tilde{a}_t) := N(\tilde{s}_t, \tilde{a}_t) + 1.$$

## 6 Main Results ( $\gamma\beta < 1$ )

In this section we summarize the main results regarding the AAA algorithm. Proofs are deferred to the next section.

Recall the definition of the mistake count (2) and the corresponding near-optimality criterion. Also, recall that  $\mathbb{S}_\epsilon$  is

the coarsest (feasible) grid with  $\Delta(s) \leq \Delta_\epsilon$ , which satisfies (13). First we present the main theorem, which provides a mistake bound of modified AAA scheme in terms of the number of cells in  $\mathbb{S}_\epsilon$ .

**Theorem 1** *Let  $\epsilon > 0$  and assume that the AAA algorithm receives an input*

$$\Delta_\epsilon = \frac{(1-\gamma)(1-\gamma\beta)}{2\alpha(\gamma+2)}\epsilon.$$

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

$$\text{PM}(\epsilon) \leq \frac{|\mathbb{S}_\epsilon| |\mathbb{A}| (2\mathcal{N} + 1)}{1-\gamma} \ln \frac{2(r_{max} - r_{min})}{\epsilon(1-\gamma)}.$$

In addition to the above theorem, we can obtain a possibly tighter mistake bound in terms of the *posterior number of cells actually used in the course of the algorithm*. In fact, the purpose of adaptive aggregation is that as time progresses, the algorithm will split cells only in the vicinity of the optimal trajectory. Therefore, the actual number of grid cells “at infinity” will be much less than  $|\mathbb{S}_\epsilon|$ . We make this more formal below.

**Definition 6** *Let  $x_0$  be the initial state and let  $N_\infty(x_0, a)$  be the number of cells in the grid  $\lim_{t \rightarrow \infty} \mathbb{S}_t(a)$ , that is*

$$N_\infty(x_0, a) \triangleq \lim_{t \rightarrow \infty} |\mathbb{S}_t(a)|. \quad (17)$$

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

We note that the limit in (17) exists and is finite, since  $|\mathbb{S}_t(a)|$  increases in  $t$ , while  $|\mathbb{S}_t(a)| \leq |\mathbb{S}_\epsilon|$  due to the enforced “stop-splitting” rule. For the same reason, we have the trivial bound  $N_\infty(x_0) \leq |\mathbb{S}_\epsilon| |\mathbb{A}|$ .

**Theorem 2** *Let  $\epsilon > 0$  and assume that the AAA algorithm receives an input  $\Delta_\epsilon$  as in Theorem 1. Then, it holds that*

$$\text{PM}(\epsilon) \leq \frac{4N_\infty(x_0)\mathcal{N}}{1-\gamma} \ln \frac{2(r_{max} - r_{min})}{\epsilon(1-\gamma)}.$$

Note that the bound of Theorem 2 becomes better than the bound of Theorem 1 if  $N_\infty(x_0) \leq \frac{1}{2} |\mathbb{S}_\epsilon| |\mathbb{A}|$ .

**Remark.** Since the action-mistake count satisfies  $\text{AM}(\epsilon) \leq \text{PM}(\epsilon)$ , the policy-mistake bounds of Theorems 1 and 2 apply also to the action-mistake.

**Discussion.** Theorem 1 implies that the mistake bound is linear in  $|\mathbb{S}_\epsilon|$ . Therefore, using equation (13), we obtain the following explicit dependence on  $\epsilon$  (ignoring the log factor):

$$\text{PM}(\epsilon) \leq C (1/\epsilon)^n |\mathbb{A}| (2\mathcal{N} + 1), \quad (18)$$

where the constant  $C$  is polynomial in  $\alpha$ ,  $1/(1-\gamma)$  and in  $1/(1-\gamma\beta)$ . Note however the exponential dependence on the dimension  $n$  of the state-space, which is an obvious artifact of the dense aggregation approach.

In the context of the posterior bound (Theorem 2), it should be noted that there is a trade-off between the choice of the count threshold  $\mathcal{N}$  and the number of cells 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 cells, 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 grid (namely,  $\mathbb{S}_\epsilon$ ) over the state space. It can be shown that the mistake bound in such case will be as in Theorem 1, with  $2\mathcal{N} + 1$  replaced by 1. Clearly, however, such an algorithm is not feasible if  $|\mathbb{S}_\epsilon|$  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 [7]). In the ideal case when the MDP assumption happens to hold true, such an algorithm will again have the mistake bound as in Theorem 1, with  $2\mathcal{N} + 1$  replaced by 1 (see for instance [12], 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 (and, in fact, in infinite mistake count).

## 7 Analysis of the AAA Scheme

Below is the outline of the analysis. First we show that the optimal value function  $V$  possesses some continuity property, which will justify the use of the smoothing operator  $\mathcal{T}_2$ . Then, we show that there exists a unique solution to equation (15), and that this solution upper bounds the optimal value  $V$ . Finally, we prove that under certain conditions on the grid, the optimal policy with respect to the UVF (equation (16)) is an  $\epsilon$ -optimal policy, which will enable us to prove a polynomial bound on the policy-mistake count of the algorithm.

For ease of exposition, throughout the analysis we write CI for the transition uncertainty set (instead of  $\text{CI}_f$ ), and denote by  $V_b \triangleq \frac{1}{1-\gamma}(r_{max} - r_{min})$  the maximal difference between two returns of any two policies. Also, recall that the proofs presented below are limited to the  $\gamma\beta < 1$  case.

### a. Continuity of the Optimal Value Function

In this subsection we show that under the continuity Assumption 2, the optimal value function is also Lipschitz continuous<sup>4</sup>.

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

$$|V(x_1) - V(x_2)| \leq \frac{\alpha}{1-\gamma\beta} d(x_1, x_2) \triangleq \omega(d(x_1, x_2)).$$

*Proof.* Fix  $x_1, x_2 \in \mathbb{X}$ . From the optimality equation (1), we have that

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

<sup>4</sup>In case  $\gamma\beta > 1$  it is Hölder continuous, see [3] for details.

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:

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

Now, since  $\gamma\beta < 1$ , we can take  $H = \infty$  in the above bound, and obtain the desired result.  $\square$

## b. The Upper Value Function

First we prove the contraction property of the upper DP operator used in the fixed point equation (15).

**Lemma 2** *The operator  $\tilde{T}$  is a contraction mapping in the  $\ell_\infty$  norm, with the contraction factor  $\gamma$ . Thus, there exists a unique solution to equation (15).*

*Proof.* Given two functions  $g_1$  and  $g_2$ , we have the following sequence of inequalities:

$$\begin{aligned} & \left| (\tilde{T}g_1)(s) - (\tilde{T}g_2)(s) \right| \\ & \leq \gamma \max_{a \in \mathbb{A}} \left| \max_{s' \in \text{CI}(s, a)} \mathcal{T}_2 g_1(s') - \max_{s' \in \text{CI}(s, a)} \mathcal{T}_2 g_2(s') \right| \\ & \leq \gamma \max_{a \in \mathbb{A}} \max_{s' \in \text{CI}(s, a)} |\mathcal{T}_2 g_1(s') - \mathcal{T}_2 g_2(s')| \\ & \leq \gamma \max_{s' \in \mathbb{S}_t} |\mathcal{T}_2 g_1(s') - \mathcal{T}_2 g_2(s')|. \end{aligned}$$

Now, since

$$\begin{aligned} & |\mathcal{T}_2 g_1(s') - \mathcal{T}_2 g_2(s')| \\ & = \left| \min_{s'' \in \mathbb{S}_t} \{g_1(s'') + \omega(\Delta(s') + d_b(s', s''))\} - \min_{s'' \in \mathbb{S}_t} \{g_2(s'') + \omega(\Delta(s') + d_b(s', s''))\} \right| \\ & \leq \max_{s'' \in \mathbb{S}_t} |g_1(s'') - g_2(s'')| = \|g_1 - g_2\|_\infty, \end{aligned}$$

it follows that  $\left| (\tilde{T}g_1)(s) - (\tilde{T}g_2)(s) \right| \leq \gamma \|g_1 - g_2\|_\infty$  for all  $s \in \mathbb{S}_t$ . Hence,  $\left\| \tilde{T}g_1 - \tilde{T}g_2 \right\|_\infty \leq \gamma \|g_1 - g_2\|_\infty$ , which proves the result.  $\square$

We will need the following property of the smoothing operator  $\mathcal{T}_2$ .

**Lemma 3** *If  $g_1 : \mathbb{S}_t \rightarrow \mathbb{R}$  is an upper bound on the value function (that is,  $g_1(s) \geq V(x)$  for all  $s \in \mathbb{S}_t$  and  $x \in s$ ), then so is  $g_2 \triangleq \mathcal{T}_2 g_1$ .*

*Proof.* For given  $s \in \mathbb{S}_t$ , let  $s^*$  be the cell that achieves the minimum in the smoothing operator  $\mathcal{T}_2$ :

$$s^* = \operatorname{argmin}_{s' \in \mathbb{S}_t} \{g_1(s') + \omega(\Delta(s) + d_b(s, s'))\}.$$

If  $s = s^*$ , then by definition of the biased distance (5) we have that  $d_b(s, s^*) = -\Delta(s)$ , implying that

$$\omega(\Delta(s) + d_b(s, s^*)) = \omega(0) = 0.$$

Thus,  $g_2(s) = g_1(s) \geq V(x)$  for all  $x \in s$ . Otherwise, let<sup>5</sup>  $x_{\min} \in \bar{s}$  and  $x_{\min}^* \in \bar{s}^*$  be such that  $d_b(s, s^*) = d(x_{\min}, x_{\min}^*)$ . We have that

$$\begin{aligned} g_2(s) & \triangleq g_1(s^*) + \omega(\Delta(s) + d_b(s, s^*)) \\ & \geq V(x_{\min}^*) + \omega(\Delta(s) + d_b(s, s^*)) \\ & \geq V(x), \end{aligned}$$

where the first inequality follows by hypothesis for the state  $x_{\min}^* \in \bar{s}^*$ , and the second inequality holds for every  $x \in s$  by Lemma 1, since

$$d(x, x_{\min}^*) \leq d(x_{\min}, x_{\min}^*) + d(x, x_{\min}) \leq d_b(s, s^*) + \Delta(s).$$

$\square$

**Lemma 4** *The UVF  $\tilde{V}_t$  is indeed an upper bound on the optimal value function. That is, at every time  $t$ , we have that  $\tilde{V}_t(s) \geq V(x)$ ,  $\forall s \in \mathbb{S}_t, \forall x \in s$ .*

*Proof.* Since, by Lemma 2,  $\tilde{T}$  is a contraction operator, we can prove the claim by induction on the steps of value iteration. For the base case, let  $\tilde{V}^0(s) \equiv V_{\max} \geq V(x), \forall x \in \mathbb{X}$ . 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  $\tilde{r}(s, a)$ , that for all  $s \in \mathbb{S}_t$  and  $x \in s$ ,

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

where  $x_s$  is a sample point in  $s$ . Also, by Assumption 2 and by the definition of  $\text{CI}(s, a)$ , it follows for any  $x \in s$ , that  $f(x, a) \in s'$ , with  $s' \in \text{CI}(s, a)$ . Thus,

$$\max_{s' \in \text{CI}(s, a)} \mathcal{T}_2 \tilde{V}^n(s') \geq \mathcal{T}_2 \tilde{V}^n(s' : f(x, a) \in s') \geq V(f(x, a)),$$

where the last inequality follows by the induction assumption and Lemma 3. Therefore, we have

$$\begin{aligned} \tilde{V}^{n+1}(s) & = \max_{a \in \mathbb{A}} \left\{ \tilde{r}(s, a) + \gamma \max_{s' \in \text{CI}(s, a)} \mathcal{T}_2 \tilde{V}^n(s') \right\} \\ & \geq \max_{a \in \mathbb{A}} \{r(x, a) + \gamma V(f(x, a))\} \\ & = V(x). \end{aligned}$$

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

## c. Near-Optimality of the UVF Optimal Policy

In this section we provide a sufficient condition on the grid, 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  $\epsilon$ -close to the UVF:  $\tilde{V}_t(s) - J_M^{\mathcal{A}_t}(x) \leq \epsilon$ , for a given  $s \in \mathbb{S}_t$  and all  $x \in s$ . 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  $\epsilon$ -optimal policy.

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

<sup>5</sup> $\bar{A}$  denotes the closure of a set  $A$ .

**Definition 7 (Known Pairs)** At any time  $t$ , define the set of actually known cell–action pairs:

$$AK_t \triangleq \{(s, a) \in \mathbb{S}_t(a) \times \mathbb{A} : \Delta(s) \leq \Delta_\epsilon, (s, a) \text{ was sampled}\}.$$

Also, define the set of virtually known cell–action pairs:

$$VK_t \triangleq \{(s, a) \in \mathbb{S}_t(a) \times \mathbb{A} : \Delta(s) \leq \Delta_\epsilon\}.$$

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 defined to be

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

**Definition 9 (Escape Event)** At any time  $t$ , define the actual escape event from a given starting cell  $s \in \mathbb{S}_t$ :

$$AE_t(s) \triangleq \left\{ \begin{array}{l} \text{In a path starting from cell } s \text{ and following} \\ \mathcal{A}_t \text{ for } T_{\epsilon/2} \text{ steps in } M, \text{ an actually visited} \\ \text{pair } (s'_t, a_t) \text{ not in } AK_t \text{ is encountered.} \end{array} \right\}$$

Also, define the virtual escape event from a given starting cell  $s \in \mathbb{S}_t$ :

$$VE_t(s) \triangleq \left\{ \begin{array}{l} \text{In a path starting from cell } s \text{ and following} \\ \mathcal{A}_t \text{ for } T_{\epsilon/2} \text{ steps in } M, \text{ a virtually visited} \\ \text{pair } (\tilde{s}_t, \tilde{a}_t) \text{ not in } VK_t \text{ is encountered.} \end{array} \right\}$$

Finally, the escape event is defined as

$$E_t(s) = AE_t(s) \cup VE_t(s).$$

**Definition 10 (Episode)** An episode is a maximal period of time  $[t_0, t_1]$ , in which:

- (i) All actually and virtually visited cell–action pairs till time  $t_1 - 1$  satisfy

$$\Delta(s'_t) \leq \Delta_\epsilon, \Delta(\tilde{s}_t) \leq \Delta_\epsilon,$$

and in addition,  $\{(s'_t, a_t)\}_{t=t_0}^{t_1-1}$  were previously sampled (that is, were 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 cell size threshold parameter  $\Delta_\epsilon$ . The next lemma formulates the condition on  $\Delta_\epsilon$  which will imply that the execution of the algorithm's implemented policy  $\mathcal{A}_t$  from time  $t$  will obtain a return which is  $\epsilon$ -close to the UVF.

**Lemma 5** Let  $\epsilon > 0$  be given and assume that

$$\Delta_\epsilon = \frac{(1-\gamma)(1-\gamma\beta)}{2\alpha(\gamma+2)}\epsilon. \quad (19)$$

Then,

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

holds for all  $t$ ,  $s \in \mathbb{S}_t$  and  $x \in s$ .

*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  $M$ . We will use the notation of visited grid cells specified in Definition 5, with  $t_1 = t_0 + T_{\epsilon/2} - 1$ . Now, we have two mutually exclusive cases: (a) For all  $(s'_t, a_t)$  it holds that  $\Delta(s'_t) \leq \Delta_\epsilon$  and  $(s'_t, a_t)$  was sampled, and for all  $s_t^*$  it holds that  $\Delta(s_t^*) \leq \Delta_\epsilon$ . (b) There exists at least one  $t \in [t_0, t_1]$  such that the above condition regarding either  $s'_t$  or  $s_t^*$  does not hold.

The case (b) above is easy – if it happens, we have that either a pair  $(s', a)$  not in  $AK_t$  is encountered, or a virtually visited cell  $s^*$  with  $\Delta(s^*) > \Delta_\epsilon$  is encountered. In the latter case, since the corresponding virtually visited cell-action pair  $(\tilde{s}, \tilde{a})$  satisfies  $s^* \subseteq \tilde{s}$ , we have that this pair is not in  $VK_t$ . Thus, the escape event  $E_{t_0}(s)$  occurred during the execution of  $\mathcal{A}_{t_0}$  for  $T_{\epsilon/2}$  time-steps, which is expressed by the  $\mathbb{I}\{E_{t_0}(s)\}V_b$  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, and thus the algorithm's policy remains unchanged and it is the stationary policy  $\pi_{t_0}$ . For simplicity, assume  $t_0 = 0$ , write  $\pi$  for  $\pi_0$  and  $\tilde{V}$  for  $\tilde{V}_0$ , and recall that  $\pi$  is the greedy policy with respect to  $\tilde{V}$  (equation (16)). Thus,  $\tilde{V}(s_0) = \tilde{r}(s'_0, a_0) + \gamma \mathcal{T}_2 \tilde{V}(s_1^*)$ . 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

$$\begin{aligned} & \tilde{V}(s_0) - J_M^\pi(x_0) \\ & \leq 2\alpha\Delta(s'_0) + \gamma \left( \mathcal{T}_2 \tilde{V}(s_1^*) - J_M^\pi(x_1) \right) \\ & \leq 2\alpha\Delta(s'_0) \\ & \quad + \gamma \left( \tilde{V}(s_1) + \omega(\Delta(s_1^*) + d_b(s_1^*, s_1)) - J_M^\pi(x_1) \right) \\ & \leq 2\alpha\Delta(s'_0) \\ & \quad + \gamma\omega(\Delta(s_1^*) + 2\beta\Delta(s'_0)) + \gamma \left( \tilde{V}(s_1) - J_M^\pi(x_1) \right). \end{aligned}$$

The first inequality follows by the definition of virtually visited cells, and by the continuity assumption on the reward, since

$$\begin{aligned} \tilde{r}(s'_0, a_0) - r(x_0, a_0) &= r(x', a_0) + \alpha\Delta(s'_0) - r(x_0, a_0) \\ &\leq 2\alpha\Delta(s'_0), \end{aligned}$$

where  $x'$  is the sample that was received for this cell-action pair. The second inequality follows by the definition of the smoothing operator (Definition 3). Note that  $s_1 \in \mathbb{S}_1$  by its definition (cf. Definition 5). Finally, the third inequality follows since both  $s_1^*$  and  $s_1$  are in  $\text{CI}(s'_0, a_0)$ , having

$$\begin{aligned} d_b(s_1^*, s_1) &\leq d_b(s_1^*, \hat{f}(s'_0, a_0)) + d_b(\hat{f}(s'_0, a_0), s_1) \\ &\leq \beta\Delta(s'_0) + \beta\Delta(s'_0). \end{aligned}$$

Thus, proceeding iteratively, we obtain the following bound:

$$\begin{aligned} & \tilde{V}(s_0) - J_M^\pi(x_0) \\ & \quad \stackrel{T_{\epsilon/2}-1}{=} \\ & \leq \sum_{t=0}^{T_{\epsilon/2}-1} \gamma^t \left[ 2\alpha\Delta(s'_t) + \gamma\omega(\Delta(s_{t+1}^*) + 2\beta\Delta(s'_t)) \right] \\ & \quad + \gamma^{T_{\epsilon/2}} V_b. \end{aligned}$$

By the definition of  $T_{\epsilon/2}$  (Definition 8), we have  $\gamma^{T_{\epsilon/2}} V_b \leq \frac{\epsilon}{2}$ . Now, we have to check that the condition (19) of the lemma regarding  $\Delta_\epsilon$ , implies that

$$\sum_{t=0}^{T_{\epsilon/2}-1} \gamma^t [2\alpha\Delta(s'_t) + \gamma\omega(\Delta(s_{t+1}^*) + 2\beta\Delta(s'_t))] \leq \frac{\epsilon}{2}.$$

Indeed,

$$\begin{aligned} & \sum_{t=0}^{T_{\epsilon/2}-1} \gamma^t [2\alpha\Delta(s'_t) + \gamma\omega(\Delta(s_{t+1}^*) + 2\beta\Delta(s'_t))] \\ & \leq \sum_{t=0}^{T_{\epsilon/2}-1} \gamma^t [2\alpha\Delta_\epsilon + \gamma\omega(\Delta_\epsilon + 2\beta\Delta_\epsilon)] \\ & \leq \sum_{t=0}^{\infty} \gamma^t [2\alpha\Delta_\epsilon + \gamma\omega(\Delta_\epsilon + 2\beta\Delta_\epsilon)] \\ & = \frac{1}{1-\gamma} \left[ 2\alpha\Delta_\epsilon + \gamma \frac{\alpha}{1-\gamma\beta} (1+2\beta)\Delta_\epsilon \right] \\ & = \frac{2\alpha + \gamma\alpha}{(1-\gamma)(1-\gamma\beta)} \Delta_\epsilon = \frac{\epsilon}{2}, \end{aligned}$$

where the first inequality follows since we are addressing case (a), in which all actually and virtually visited cells are smaller than  $\Delta_\epsilon$ , the second inequality holds by taking the infinite sum instead of the finite one, the first equality follows by the definition of  $\omega$  (see Definition 3), and the last equality follows by the hypothesis (19) of the Lemma. This completes the proof of the Lemma.  $\square$

#### d. Number of Cells in $\epsilon$ -Optimal Grid

Before proving the mistake bounds, we provide an upper bound on the number of cells  $N_\epsilon = |\mathbb{S}_\epsilon|$ .

**Lemma 6** *For a fixed feasible splitting scheme, with parameters  $c_{split}$  and  $\lambda$ , and a single initial grid  $\mathbb{S}_0$ , we have that*

$$N_\epsilon \leq |\mathbb{S}_0| c_{split} \left( \frac{\Delta_{max}}{\Delta_\epsilon} \right)^{\log_{1/\lambda}(c_{split})}.$$

*Proof.* For every  $s \in \mathbb{S}_0$ , consider performing  $k(s)$  splits iteratively, such that at each iteration we obtain new  $c_{split}$  cells instead of the original one. It follows that after  $k(s)$  such splits, the size of a split cell  $s' \subseteq s$  satisfies  $\Delta(s') \leq \lambda^{k(s)} \Delta(s)$ . In addition, the number of cells in the grid that contains all such cells  $s'$  is

$$N = \sum_{s \in \mathbb{S}_0} c_{split}^{k(s)}. \quad (20)$$

Thus, for each  $s \in \mathbb{S}_0$ , we need to find the minimal  $k(s)$ , such that

$$\lambda^{k(s)} \Delta(s) \leq \Delta_\epsilon. \quad (21)$$

From (21), it follows that this minimal  $k(s) = k^*(s)$  satisfies

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

Substituting the last inequality in (20) yields

$$\begin{aligned} N_\epsilon &= \sum_{s \in \mathbb{S}_0} c_{split}^{k^*(s)} \\ &\leq c_{split} \sum_{s \in \mathbb{S}_0} (c_{split})^{\log_{1/\lambda} \left( \frac{\Delta(s)}{\Delta_\epsilon} \right)} \\ &= c_{split} \sum_{s \in \mathbb{S}_0} \left( \frac{\Delta(s)}{\Delta_\epsilon} \right)^{\log_{1/\lambda}(c_{split})} \\ &\leq |\mathbb{S}_0| c_{split} \left( \frac{\Delta_{max}}{\Delta_\epsilon} \right)^{\log_{1/\lambda}(c_{split})}, \end{aligned}$$

which completes the proof.  $\square$

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

#### e. Proof of Theorem 1

First, we note that the escape event  $E_t(s)$  (Definition 9) 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 cell-action pair  $(s, a)$  (either actually, or virtually), with  $s$  that is not in  $\mathbb{S}_\epsilon$ . In addition, in case of actual escape event (see Definition 9), this pair was not sampled. This fact can be interpreted as a “discovery” of new information, since every such occurrence of an “unknown” pair will lead to an increase of the count of such pair, and, eventually, to split of such a pair.

Next two lemmas show that the number of times that “actual” and “virtual” escape events can occur is bounded.

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

*Proof.* Note that any cell  $s \in \mathbb{S}_t(a)$  for any  $a$  and  $t$ , can be visited no more than  $\mathcal{N}$  times – after this number of times, the cell is split. Now think of the grid representation of the state space as a tree, with cells as leaves. The internal nodes in such tree represent the larger aggregations, 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” pair can be interpreted as a visit to *an internal node of  $\mathbb{S}_\epsilon$* . Since the counter of this pair 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  $\mathbb{S}_\epsilon$  is bounded by

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

Finally, when the algorithm encounters a leaf of  $\mathbb{S}_\epsilon$ , then only one such occurrence is sufficient in order to the desired cell to become sampled. Again, by a simple counting argument, the number of times this can occur is bounded by

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

To conclude, the number of times that an “unknown” cell-action pair 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” pair (that is, a pair not in  $AK_t$ ), there is no occurrence of the escape event  $AE_t(s)$ . (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 pair at time  $t \leq t' \leq t + T_{\epsilon/2}$ , the escape event  $AE_{t'}(s)$  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 pair at the end of the execution period 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 pair. That is, if  $AE_t(x_t)$  occurs at some time  $t$ , also  $AE_{t'}(x_{t'})$  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 pairs, all the pairs will become known,  $AE_t(s)$  can occur at most  $N_\epsilon(\mathcal{N} + 1)|\mathbb{A}|T_{\epsilon/2}$  times.  $\square$

**Lemma 8** *The number of times that  $VE_t(s)$  can occur is bounded by  $N_\epsilon|\mathbb{A}|\mathcal{N}T_{\epsilon/2}$ .*

*Proof.* The proof is similar to the proof of Lemma 7, with the difference that the virtual escape event cannot occur on the leaves of  $\mathbb{S}_\epsilon$ .  $\square$

**Lemma 9** *The number of times that the escape event  $E_t(s)$  can occur is bounded by  $N_\epsilon|\mathbb{A}|(2\mathcal{N} + 1)T_{\epsilon/2}$ .*

*Proof.* Follows by Lemmas 7 and 8 and Definition 9.  $\square$

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  $x_t (x_t \in s_t)$ . We then have that

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

where the first inequality holds by Lemma 5, and the second inequality holds by Lemma 4. However, by Lemma 9, the number of times the event  $E_t(s_t)$  can occur is bounded by  $N_\epsilon(2\mathcal{N} + 1)|\mathbb{A}|T_{\epsilon/2}$ , implying that

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

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  $C$ .  $\square$

## f. Proof of Theorem 2

Recall the definition of the posterior number  $N_\infty(x_0)$  of actually used cells in the course of the algorithm (Definition 6). We only need to prove the analogue of Lemma 9 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 grids “at infinity”, namely  $\mathbb{S}_\infty(a) = \lim_{n \rightarrow \infty} \mathbb{S}_t(a)$ ,  $a \in \mathbb{A}$ , instead of the  $\epsilon$ -optimal grid  $\mathbb{S}_\epsilon$ . First, consider the actual escape event. As previously, this event can occur on *internal nodes* of  $\mathbb{S}_\infty(a)$  no more than

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

times. The *leaves* of the tree (which are the cells of  $\mathbb{S}_\infty(a)$ ) can be classified into two groups: (a) “Small” leaves, with  $\Delta(s) \leq \Delta_\epsilon$ , and (b) “Large” leaves, with  $\Delta(s) > \Delta_\epsilon$ . On “small” leaves, only one occurrence of the escape event is possible, since such a cell becomes known (Definition 7) after one sample. On “large” leaves, there cannot be more than  $\mathcal{N}$  occurrences of the escape event – otherwise these cells would have been split. Thus, the number of times the escape event can occur on leaves is bounded by

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

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

Similarly, the virtual escape event cannot occur more than  $\sum_{a \in \mathbb{A}} 2N_\infty(x_0, a)\mathcal{N}$  times. Note that there is no difference in bounds on the number of occurrences of actual and virtual escape events, since the cells of  $\mathbb{S}_\infty(a)$  can be “large” ones, that is having  $\Delta(s) > \Delta_\epsilon$ . Thus, the virtual escape event can also happen on leaves. By the same arguments as in proof of Theorem 1, the sum of the above two bounds times the  $\epsilon/2$ -horizon time  $T_{\epsilon/2}$  is the mistake bound of the algorithm.

## 8 The Expansive Case ( $\gamma\beta > 1$ )

Our analysis above focused on case  $\gamma\beta < 1$ . When  $\gamma\beta > 1$ , the analysis becomes more involved. This can be observed for example, from the bound on the distance between optimal values of two states, presented in proof of Lemma 1:

$$|V(x_1) - V(x_2)| \leq \alpha d(x_1, x_2) \sum_{k=0}^{H-1} (\gamma\beta)^k + \gamma^H V_b. \quad (22)$$

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 (22), which grows exponentially in  $H$ , and the second term, which decays exponentially in  $H$ . A detailed treatment of this case can be found in [3] and is omitted here due to space limitations. Using the approach outlined above, it is shown that to obtain the mistake bounds of Theorems 1 and 2, the cell size threshold should be taken as  $\Delta_\epsilon = K\epsilon^\xi$ , where  $\xi \triangleq \log_{1/\gamma} \beta$ , and  $K$  is polynomial in  $\alpha, \beta, 1/(1-\gamma)$  and *exponential* in  $\xi$ ; note that  $\xi > 1$  and compare this condition to (19) in case  $\gamma\beta < 1$ . As a result, in the expansive case we obtain a worse explicit dependence of the mistake bound on  $\epsilon$ , as follows:

$$\text{PM}(\epsilon) \leq C' (1/\epsilon)^{n\xi} |\mathbb{A}| (2\mathcal{N} + 1),$$

where  $C'$  is polynomial in  $\alpha, \beta, 1/(1-\gamma)$ , and is exponential in  $\xi$ ; compare this bound to (18).

## 9 Conclusion

We presented a model-based learning algorithm that aims to solve the online, continuous state space reinforcement learning problem in deterministic domain, under continuity assumption of model parameters. We note that we did not address at all the issue of the computational complexity. The goal of the analysis was to show feasibility in the sense of sample efficiency.

Some ideas for improvement of the proposed algorithm and its analysis follow. First, it would be interesting from computational perspective, to formulate an on-line asynchronous variant, that will perform only one back-up of Value Iteration each time-step, instead of exact calculation, and analyze its performance. Also, the explicit dependence of the posterior number of cells on the number of cells needed on the *optimal trajectory* 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. Finally, evaluation of the algorithm 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 [9]). Preliminary results for this case can be found in [3]. 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. Another possible extension is to the case of continuous action space  $\mathbb{A}$ , which can be approached using aggregation, similarly to 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 more natural in such deterministic problems, as navigation in maze.

## References

- [1] A. Antos, R. Munos, and C. Szepesvari. Fitted Q-iteration in continuous action-space MDPs. In *Proceedings of Neural Information Processing Systems Conference (NIPS)*, 2007.
- [2] P. Auer and R. Ortner. Logarithmic online regret bounds for undiscounted reinforcement learning. In *Proceedings of Neural Information Processing Systems Conference (NIPS)*, 2006.
- [3] 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](http://tx.technion.ac.il/~andreyb/MSc_Thesis_final.pdf).
- [4] D. P. Bertsekas. *Dynamic Programming and Optimal Control, vol. 2*. Athena Scientific, Belmont, MA, third edition, 2007.
- [5] A. Bonarini, A. Lazaric, and M. Restelli. LEAP: an adaptive multi-resolution reinforcement learning algorithm. *To appear*.
- [6] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.
- [7] 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.
- [8] H. Chapman. Global confidence bound algorithms for the exploration-exploitation tradeoff in reinforcement learning. Master’s thesis, Technion – Israel Institute of Technology, 2007.
- [9] 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.
- [10] C. Diuk, A. L. Strehl, and M. L. Littman. A hierarchical approach to efficient reinforcement learning in deterministic domains. In *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 313–319, 2006.
- [11] G. J. Gordon. Reinforcement learning with function approximation converges to a region. In *Advances in Neural Information Processing Systems (NIPS) 12*, pages 1040–1046, 2000.
- [12] S. M. Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, Gatsby Computational Neuroscience Unit, University College London, UK, 2003.
- [13] M. Kearns and S. P. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49:209–232, 2002.
- [14] I. Menache, S. Mannor, and N. Shimkin. Q-Cut – dynamic discovery of sub-goals in reinforcement learning. In *Proceedings of the 13th European Conference on Machine Learning (ECML 2002)*, pages 187–195, 2002.
- [15] 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.
- [16] R. Munos and A. W. Moore. Variable resolution discretization in optimal control. *Machine Learning*, 49:291–323, 2002.
- [17] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994.
- [18] 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.
- [19] 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, 2006.
- [20] 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.
- [21] W. Whitt. Approximations of dynamic programs, I. *Mathematics of Operations Research*, 3(3):231–243, 1978.