An Adaptive State Aggregation Algorithm for Markov Decision Processes

Guanting Chen[1], Johann Demetrio Gaebler[1], Matt Peng[2], Chunlin Sun[1], Yinyu Ye[3]

[1]Institute for Computational and Mathematical Engineering, Stanford University
[2]Department of Electrical Engineering & Computer Sciences, University of California, Berkeley
[3]Department of Management Science and Engineering, Stanford University
{guanting, jgaeb, chunlin, yyye}@stanford.edu mattpeng@berkeley.edu

Abstract

Value iteration is a well-known method of solving Markov Decision Processes (MDPs) that is simple to implement and boasts strong theoretical convergence guarantees. However, the computational cost of value iteration quickly becomes infeasible as the size of the state space increases. Various methods have been proposed to overcome this issue for value iteration in large state and action space MDPs, often at the price, however, of generalizability and algorithmic simplicity. In this paper, we propose an intuitive algorithm for solving MDPs that reduces the cost of value iteration updates by dynamically grouping together states with similar cost-to-go values. We also prove that our algorithm converges almost surely to within $2\varepsilon/(1-\gamma)$ of the true optimal value in the ℓ^∞ norm, where γ is the discount factor and aggregated states differ by at most ε. Numerical experiments on a variety of simulated environments confirm the robustness of our algorithm and its ability to solve MDPs with much cheaper updates especially as the scale of the MDP problem increases.

Introduction

State aggregation is a long-standard approach to reduce the complexity of large-scale Markov decision processes (MDPs). The main idea of state aggregation is to define the similarity between states, and work with a system of reduced complexity size by grouping similar states into aggregate, or "mega-", states. Although there has been a variety of results on the performance of the policy using state aggregation (Li, Walsh, and Littman 2006; Van Roy 2006; Abel, Hershkowitz, and Littman 2016), a common assumption is that states are aggregated according to the similarity of their optimal cost-to-go values or state-action rewards (also konwn as Q-value). Such a scheme, which we term *pre-specified aggregation*, is generally infeasible unless the MDP is already solved.

This paper provides an online algorithm that learns to aggregate states effectively, while using the state-aggregation to facilitate the process of solving MDPs. We propose a simple and efficient state aggregation algorithm for calculating the optimal value and policy of an infinite-horizon discounted MDP that can be applied in planning problems (Baras and Borkar 2000) and generative MDP problems (Sidford et al. 2018a). The algorithm alternates between two distinct phases. During the first phase (we refer to as "global updates"), the algorithm updates the cost-to-go values for all states, trading off some efficiency to more accurately guide the cost-to-go values in the right direction; in the second phase (we refer to as "aggregate updates"), the algorithm groups together states with similar cost-to-go values based on the last sequence of global updates, and it then efficiently updates the states in each mega-state in tandem as it optimizes over the reduced space of aggregate states.

A few online algorithms that learn how to effectively aggregate states have been proposed (Ortner 2013; Duan, Ke, and Wang 2018; Sinclair, Banerjee, and Yu 2019). Compared to prior works on state aggregation that use information on the state-action value (Q-value) (Sinclair, Banerjee, and Yu 2019), transition density (Ortner 2013), and methods such as upper confidence intervals, our method is compact such that it does not require strong assumptions or extra information, and only performs updates in a manner similar to standard value iteration. Moreover, because our algorithm is value-iteration based, it features robustness and generality to a wide range of MDP problems and generative RL (Azar, Munos, and Kappen 2013). To implement our algorithm, the inputs needed for state aggregation are only the current cost-to-go values and the parameter ε, which bounds the difference between (current) cost to go values of states within a given mega-state.

Contribution

Our contribution is a feasible online algorithm for learning aggregate states and cost-to-go values that requires no extra information beyond that required for standard value iteration. We showcase in our experimental results that our method provides significantly faster convergence than standard value iteration especially for problems with larger state and action spaces. We also provide theoretical guarantees for the convergence, accuracy, and convergence rate of our algorithm.

Our work provides insights to the current literature in the following ways. Compared to the literature in pre-specified aggregation (Li, Walsh, and Littman 2006; Van Roy 2006; Abel, Hershkowitz, and Littman 2016), where the authors develop convergence properties for different function approximators, we also provide our convergence results, while

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

proposing online algorithm that performs state-aggregation. In the literature of online aggregation algorithms (Bertsekas, Castanon et al. 1988; Ortner 2013; Duan, Ke, and Wang 2018; Sinclair, Banerjee, and Yu 2019), our method features a more compact approach that requires less information on inputs and assumptions. The simplicity and robustness of our novel state aggregation algorithm demonstrates its utility and general applicability in comparison to existing approaches for solving large MDPs.

Related literature

When a pre-spcified aggregation is given, Tsitsiklis and Van Roy (1996) and Van Roy (2006) give performance bounds on the aggregated system and propose variants of value iteration. There are also a variety of ways to perform state aggregation based on different criteria. Ferns, Panangaden, and Precup (2012) and Dean, Givan, and Leach (1997) analyze partitioning the state space by grouping states whose transition model and reward function are close. McCallum (1997) proposes aggregate states that have the same optimal action and similar Q-values for these actions. Jong and Stone (2005) develops aggregation techniques such that states are aggregated if they have the same optimal action. We refer the readers to Li, Walsh, and Littman (2006), Abel, Hershkowitz, and Littman (2016), Abel et al. (2020) for a more comprehensive survey on this topic.

Dynamic learning of the aggregate states has also been studied more generally in MDP and reinforcement learning (RL) settings. Hostetler, Fern, and Dietterich (2014) proposes a class of Q-value–based state aggregations and applies them to Monte Carlo tree search. Slivkins (2011) uses data-driven discretization to adaptively discretize state and action space in a contextual bandit setting. Ortner (2013) develops an algorithm for learning state aggregation in an online setting by leveraging confidence intervals. Sinclair, Banerjee, and Yu (2019) designs a Q-learning algorithm based on data-driven adaptive discretization of the state-action space. For more state abstraction techniques see Baras and Borkar (2000), Dean, Givan, and Leach (1997), Jiang, Singh, and Lewis (2014), and Abel et al. (2019).

Our adaptive state-aggregated value iteration algorithm is also related to the so-called aggregation-disaggregation method used to accelerate the convergence of value iterations (Bertsekas, Castanon et al. 1988; Schweitzer, Puterman, and Kindle 1985; Mendelssohn 1982; Bertsekas 2018) in policy evaluation, i.e., to evaluate the value function of a policy. Among those works, that of Bertsekas, Castanon et al. (1988) is closest to our approach. Assuming the underlying Markov processes to be ergodic, the authors propose to group states based on Bellman residuals in between runs of value iteration. They also allow states to be aggregated or disaggregated at every abstraction step. Our work is different from Bertsekas, Castanon et al. (1988) such that in Bertsekas, Castanon et al. (1988) the convergence of the Bellman residual based state-aggregation schemes is analytically intractable, while our algorithm converges with less assumptions. Aside from state aggregation, a variety of other methods have been studied to accelerate value iteration. Herzberg and Yechiali (1994) propose iterative algorithms based on a one-step look-ahead approach. Shlakhter et al. (2010) combine the so called "projective operator" with value iteration and achieve better efficiency. Anderson (1965), Fang and Saad (2009), and Zhang, O'Donoghue, and Boyd (2020) analyze the Anderson mixing approach to speed up the convergence of fixed-point problems.

Preliminaries

Markov decision process

We consider an infinite-horizon Markov Decision Process $M = (\mathcal{S}, \mathcal{A}, P, r, \gamma, \rho)$, consisting of: a finite state space $\mathcal{S}$; a finite action space $\mathcal{A}$; the probability transition model P, where $P(s'|s,a)$ denotes the probability of transitioning to state s' conditioned on the state-action pair (s,a); the immediate cost function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which denotes the immediate cost—or reward—obtained from a particular state-action pair; the discount factor $\gamma \in [0,1)$; and the initial distribution over states in $\mathcal{S}$, which we denote by ρ.

A policy $\pi : \mathcal{S} \to \mathcal{A}$ specifies the agent's action based on the current state, either deterministically or stochastically. A policy induces a distribution over the trajectories $\tau = (s_t, a_t, r_t)_{t=0}^{\infty}$, where $s_0 \sim \rho$, and $a_t \sim \pi(\cdot|s_t)$ and $s_{t+1} \sim P(\cdot|s_t, a_t)$ for $t \geq 0$.

A value function $\boldsymbol{V} : \mathcal{S} \to \mathbb{R}^{|\mathcal{S}|}$ assigns a value to each state; as $|\mathcal{S}| < \infty$, $\boldsymbol{V}$ can also be represented by a finite-length vector $(V(1), ..., V(|\mathcal{S}|))^\top \in \mathbb{R}^{|\mathcal{S}|}$. (In this paper, we view all vector as vector functions mapping from the index to the corresponding entry.) Moreover, each policy π is associated with a value function $\boldsymbol{V}^\pi : \mathcal{S} \to \mathbb{R}$, which is defined to be the discounted sum of future rewards starting at s and with policy π:

$$\boldsymbol{V}^\pi(s) := \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | \pi, s_0 = s\right].$$

As noted above, we represent both the value function corresponding to the policy π and the value vector $(V^\pi(1), ..., V^\pi(|\mathcal{S}|))^\top$ as $\boldsymbol{V}^\pi$.

For each state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ belonging to state s, let $\boldsymbol{P}_{s,a} \in \mathbb{R}^{|\mathcal{S}|}$ denote the vector of transition probabilities resulting from taking the action a in state s. We call a policy π *greedy* with respect to a given value $\boldsymbol{V} \in \mathbb{R}^{|\mathcal{S}|}$ if

$$\pi(s) \in \underset{a \in \mathcal{A}}{\arg\min} \left(r(s,a) + \gamma \cdot \boldsymbol{P}_{s,a}^\top \boldsymbol{V}\right).$$

We define $\boldsymbol{T} : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^{|\mathcal{S}|}$ to be the dynamic programming operator such that $(\boldsymbol{T}\boldsymbol{V})(s) = T_s(\boldsymbol{V})$ where

$$T_s(\boldsymbol{V}) = \min_{a \in \mathcal{A}} \left(r(s,a) + \gamma \cdot \boldsymbol{P}_{s,a}^\top \boldsymbol{V}\right).$$

The optimal value function $\boldsymbol{V}^*$ is the unique solution of the equation $\boldsymbol{V}^* = \boldsymbol{T}\boldsymbol{V}^*$. A common approach to find $\boldsymbol{V}^*$ is value iteration, which, given an initial guess $\boldsymbol{V}_0$, generates a sequence of value functions $\{\boldsymbol{V}_t\}_{t=0}^{\infty}$ such that $\boldsymbol{V}_{t+1} = \boldsymbol{T}\boldsymbol{V}_t$. The sequence $\{\boldsymbol{V}_t\}_{t=0}^{\infty}$ converges to V^* as t goes to $+\infty$ (Bellman 1957).

State aggregation

The state space of MDPs can be very large. State aggregation divides the state space $\mathcal{S}$ into K subsets and views each collection of states as a mega-state. Then, the value function generated by the mega-states can be used to approximate the optimal value $\boldsymbol{V}^*$.

To represent a state aggregation, we define the matrix $\boldsymbol{\Phi} \in \mathbb{R}^{|\mathcal{S}| \times K}$. We set $\phi_{i,j} = 1$ if state i is in the j-th mega-state, and let $\phi_{i,j} = 0$ otherwise; i.e., column j of $\boldsymbol{\Phi}$ indicates whether each state belongs to mega-state j. The state-reduction matrix $\boldsymbol{\Phi}$ also induces a partition $\{S_i\}_{i=1}^K$ on $\mathcal{S}$, i.e., $\mathcal{S} = \bigcup_{i=1}^K S_i$ and $S_i \cap S_j = \emptyset$ for $i \neq j$. Denote by $\boldsymbol{W} \in \mathbb{R}^K$ the cost-to-go value function for the aggregated state, and note that the current value of $\boldsymbol{W}$ induces a value function $\tilde{\boldsymbol{V}}(\boldsymbol{W}) \in \mathbb{R}^{|\mathcal{S}|}$ on the original state space, where

$$\tilde{\boldsymbol{V}}(s, \boldsymbol{W}) = \boldsymbol{W}(j), \quad \text{for } s \in S_j.$$

Algorithm design

In this section, we first introduce a state aggregation algorithm which assumes knowledge of the optimal value function. The algorithm is proposed in Tsitsiklis and Van Roy (1996) which also provides the corresponding convergence result. Based on existing theory, we design our adaptive algorithm and discuss its convergence properties.

A pre-speficied aggregation algorithm

Given a pre-specified aggregation, we seek the value function $\boldsymbol{W}$ for the aggregated states such that

$$\|\tilde{\boldsymbol{V}}(\boldsymbol{W}) - \boldsymbol{V}^*\|_\infty = O(\|\boldsymbol{e}\|_\infty), \tag{1}$$

where $\boldsymbol{e} = (e_1, ..., e_K)^\top$ and $e_j = \max_{s_1, s_2 \in S_j} |V^*(s_1) - V^*(s_2)|$ for $j = 1, ..., K$. Intuitively, Eq. (1) justifies our approach of aggregating states that have similar optimal cost-to-go values. We then state the algorithm that will converge to the correct cost-to-go values while satisfying Eq. (1).

Algorithm 1: Random Value Iteration with Aggregation

```
1: Input: $\boldsymbol{P}$, $r$, $\gamma$, $\boldsymbol{\Phi}$, $\{\alpha_t\}_{t=1}^\infty$
2: Initialize $\boldsymbol{W}_0 = \boldsymbol{0}$
3: for $t = 1, ..., n$ do
4:     for $j = 1, ..., K$ do
5:         Sample state $s$ uniformly from set $S_j$
6:         $W_{t+1}(j) = (1 - \alpha_t) W_t(j) + \alpha_t T_s \tilde{\boldsymbol{V}}(\boldsymbol{W}_t)$
7:     end for
8: end for
9: Output: $\tilde{\boldsymbol{V}}_n$
```

Algorithm 1 takes a similar form in Stochastic Approximation (Robbins and Monro 1951; Wasan 2004), and will converge almost surely to a unique cost-to-go value. Here α_t is the step size of the learning algorithm; by taking, e.g., $\alpha_t = 1$, we recover the formula of value iteration. The following convergence result is proved in Tsitsiklis and Van Roy (1996).

Proposition 1 (Theorem 1, (Tsitsiklis and Van Roy 1996)) *When $\sum_{t=1}^\infty \alpha_t = \infty$ and $\sum_{t=1}^\infty \alpha_t^2 < \infty$, $\{\boldsymbol{W}_t\}_{t=1}^\infty$ in Line 6 of Algorithm 1 will converge almost surely to $\boldsymbol{W}^*$ entry-wise, where $\boldsymbol{W}^*$ is the solution of*

$$W^*(j) = \frac{1}{|S_j|} \sum_{s \in S_j} T_s \tilde{\boldsymbol{V}}(\boldsymbol{W}^*). \tag{2}$$

Define $\pi^{\boldsymbol{W}^*}$ to be the greedy policy with respect to $\tilde{\boldsymbol{V}}(\boldsymbol{W}^*)$. Then, we have, morevoer, that

$$\begin{aligned} \|\tilde{\boldsymbol{V}}(\boldsymbol{W}^*) - \boldsymbol{V}^*\|_\infty &\leq \frac{\|\boldsymbol{e}\|_\infty}{1-\gamma}, \\ \|\boldsymbol{V}^{\pi^{\boldsymbol{W}^*}} - \boldsymbol{V}^*\|_\infty &\leq \frac{2\gamma\|\boldsymbol{e}\|_\infty}{(1-\gamma)^2}, \end{aligned} \tag{3}$$

where $\boldsymbol{V}^{\pi^{\boldsymbol{W}^*}}$ is the value function associated with policy $\pi^{\boldsymbol{W}^*}$.

Proposition 1 states that if we are able to partition the state space such that the maximum difference of the optimal value function within each mega-state is small, the value function produced by Algorithm 1 can approximate the optimal value up to $\frac{\|\boldsymbol{e}\|_\infty}{1-\gamma}$, and the policy associated with the approximated value function will also be close to the optimal policy.

Value iteration with state aggregation

In order to generate an efficient approximation, Proposition 1 requires a pre-specified aggregation scheme such that $\max_{s_1, s_2 \in S_i} |V^*(s_1) - V^*(s_2)|$ is small for every i to guarantee the appropriate level of convergence for Algorithm 1. Without knowing $\boldsymbol{V}^*$, is it still possible to control the approximation error? In this section we answer in the affirmative by introducing an adaptive state aggregation scheme that learns the correct state aggregations online as it learns the true cost-to-go values.

Given the current cost-to-go value vector $V \in \mathbb{R}^{|\mathcal{S}|}$, let $b_1 = \min_{s \in \mathcal{S}} V(s)$, let $b_2 = \max_{s \in \mathcal{S}} V(s)$. Group the cost-to-go values among disjoint subintervals of the form $\lceil (b_2 - b_1)/\varepsilon \rceil$. Next, let $\Delta = (b_2 - b_1)/\varepsilon$, and let S_j to be the j-th mega-state, which contains all the states whose current estimated cost-to-go value falls in the interval $[b_1 + (j-1)\varepsilon, b_1 + j\varepsilon)$. Grouping the states in this way reduces the state size from $|\mathcal{S}|$ states to at most $\lceil (b_2 - b_1)/\varepsilon \rceil$ mega-states. See Algorithm 2 for further details.

Without the knowledge of $\boldsymbol{V}^*$ in advance, one must periodically perform value iteration on $\mathcal{S}$ to learn the correct aggregation to help with adapting the aggregation scheme. As a result, our algorithm alternates between two phases: in the global update phase the algorithm performs value iteration on $\mathcal{S}$; in the aggregated update phase, the algorithm starts to group together states with similar cost-to-go values based on the result of the last global update, and then performs aggregated updates as in Algorithm 1.

We denote by $\{\mathcal{A}_i\}_{i=1}^\infty$ the intervals of iterations in which the algorithm performs state-aggregated updates, and we denote by $\{\mathcal{B}_i\}_{i=1}^\infty$ the intervals of iterations in which the algorithm performs global update. As a consequence, $b < a$ for

Algorithm 2: Value-based Aggregation

```
1: Input: $\varepsilon$, $\boldsymbol{V} = (V(1), ..., V(|\mathcal{S}|))^\top$
2: $b_1 = \min_{s \in |\mathcal{S}|} V(s)$, $b_2 = \max_{s \in |\mathcal{S}|} V(s)$, $\Delta = (b_2 - b_1)/\varepsilon$
3: for $i = 1, ..., \lceil \Delta \rceil$ do
4:     $\hat{S}_i = \{s | V(s) \in [b_1 + (i-1)\varepsilon, b_1 + i\varepsilon)\}$, $\hat{W}(i) = b_1 + (i - \frac{1}{2})\varepsilon$
5: end for
6: Delete the empty sets in $\{\hat{S}_i\}_{i=1}^{\lceil \Delta \rceil}$ while keep the same order, and define the modified partition to be $\{S_i\}_{i=1}^{K}$, where $K$ is the cardinally of the modified set of mega-states. Modify $\hat{W}$ and generate $W \in \mathbb{R}^K$ the similar way.
7: Return $\{S_i\}_{i=1}^{K}$ and $W$.
```

any $a \in \mathcal{A}_i$ and $b \in \mathcal{B}_i$; likewise, $a < b$ for any $a \in \mathcal{A}_i$ and $b \in \mathcal{B}_{i+1}$.

We then present our adaptive algorithm. For a prespeficied number of iterations n, the time horizon $[1, n)$ is divided into intervals of the form $\mathcal{B}_1, \mathcal{A}_1, \mathcal{B}_2, \mathcal{A}_2, \ldots$. Every time the algorithm exits an interval of global updates $\mathcal{B}_i$, it runs Algorithm 2 based on the current cost-to-go value and the parameter ε, using the output of Algorithm 2 for the current state aggregation and cost-to-go values for $\mathcal{A}_i$. Similarly, every time the algorithm exits an interval of state-aggregated updates $\mathcal{A}_i$, it sets $\tilde{\boldsymbol{V}}(\boldsymbol{W})$, where $\boldsymbol{W}$ is the current cost-to-go value for the aggregated space, as the initial cost-to-go value for the subsequent interval of global iterations.

Convergence

From Proposition 1, if we fix the state-aggregation parameter ε, even with perfect information, state aggregated value iteration will generate an approximation of the cost-to-go values with ℓ^∞ error bounded by $\varepsilon/(1-\gamma)$. This bound is sharp, as shown in (Tsitsiklis and Van Roy 1996). Such error is negligible in the early phase of the algorithm, but the error would accumulate in the later phase of the algorithm and prevent the algorithm from converging to the optimal value. As a result, it is not desirable for $\limsup |\mathcal{A}_t| \to \infty$.[1]

We state asymptotic convergence results for Algorithm 3; proofs can be found in the supplementary materials. For the remainder of the paper, by a slight abuse of notation, we denote by $\boldsymbol{V}_t$ the current cost-to-go value at iteration t. More specifically, if the current algorithm is in phase $\mathcal{B}_i$, $\boldsymbol{V}_t$ is the updated cost-to-go value for global value iteration. If the algorithm is in phase $\mathcal{A}_i$, $\boldsymbol{V}_t$ represents $\tilde{\boldsymbol{V}}(\boldsymbol{W}_t)$.

Theorem 1 *If* $\limsup \alpha_t \to 0$, $\limsup_{t\to\infty} |\mathcal{A}_i| < \infty$ *and* $\liminf_{t\to\infty} |\mathcal{B}_i| > 0$, *we have*

$$\limsup_{t\to\infty} \|\boldsymbol{V}_t - \boldsymbol{V}^*\|_\infty \leq \frac{2\varepsilon}{1-\gamma}.$$

[1]By setting ε adaptively, one might achieve better complexity and error bound by setting $\limsup |\mathcal{A}_t| \to \infty$; however, adaptively choosing ε lies beyond the scope of this work.

Algorithm 3: Value Iteration with Adaptive Aggregation

```
1: Input: $P$, $r$, $\varepsilon$, $\gamma$, $\{\alpha_t\}_{t=1}^\infty$, $\{\mathcal{A}_i\}_{i=1}^\infty$, $\{\mathcal{B}_i\}_{i=1}^\infty$
2: Initialize $W_0 = \mathbf{0}$, $V_1 = \mathbf{0}$, $t_{sa} = 1$
3: for $t = 1, ..., n$ do
4:     if $t \in \mathcal{B}_i$ then
5:         if $t = \min\{\mathcal{B}_i\}$ then
6:             $\boldsymbol{V}_{t-1} = \tilde{\boldsymbol{V}}(\boldsymbol{W}_{t-1})$.
7:         end if
8:         for $j = 1, ..., |\mathcal{S}|$ do
9:             $V_t(j) = T_j \boldsymbol{V}_{t-1}$.
10:        end for
11:    else
12:        Find current $i$ s.t. $t \in \mathcal{A}_i$
13:        if $t = \min\{\mathcal{A}_i\}$ then
14:            Define $\{S_i\}_{i=1}^K$ and $\boldsymbol{W}_t$ to be the output of Algorithm 2 with input $\varepsilon$, $\boldsymbol{V}_{t-1}$.
15:        end if
16:        for $j = 1, ..., K$ do
17:            Sample state $s$ uniformly from set $S_j$. Update
               $$W_t(j) = (1 - \alpha_{t_{sa}}) W_{t-1}(j) + \alpha_{t_{sa}} T_s \tilde{\boldsymbol{V}}(\boldsymbol{W}_{t-1}) \quad (4)$$
18:        end for
19:        $t_{sa} = t_{sa} + 1$
20:    end if
21: end for
22: if $n \in \mathcal{B}_i$ then
23:     return $\boldsymbol{V}_n$.
24: end if
25: return $\tilde{\boldsymbol{V}}(\boldsymbol{W}_n)$.
```

Notice that the result of Theorem 1 is consistent with Proposition 1: due to state aggregation we suffer from the same $\frac{1}{1-\gamma}$ for the error bound. However, our algorithm maintains the same order of error bound without knowing the optimal value function $\boldsymbol{V}^*$.

We also identify the existence of "stable field" for our algorithm, and we prove that under some specific choice of the learning rate α_t, with probability one the value function will stay within the stable field.

Proposition 2 *If* $\alpha_t < \frac{\varepsilon}{(1-\gamma)\sup_i |\mathcal{A}_i|}$ *for all t, we have that after* $O\left(\max\left\{1, \frac{\log(\varepsilon)}{\log(\gamma)}\right\}\right)$ *iterations, with probability one the estimated approximation* $\boldsymbol{V}_t$ *satisfies*

$$\|\boldsymbol{V}_t - \boldsymbol{V}^*\|_\infty \leq \frac{3\varepsilon}{1-\gamma}$$

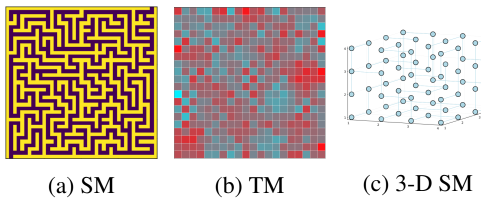
for any choice of initialization $\boldsymbol{V}_0 \in \left[0, \frac{1}{1-\gamma}\right]^{|\mathcal{S}|}$.

Proposition 3 *For* $\beta > 0$, *if* $\alpha_t \leq t^{-\beta}$ *, after* $O(\frac{\log(\varepsilon)}{\log(\gamma)} + (1-\gamma)^{-\frac{1}{\beta}}\varepsilon^{-\frac{1}{\beta}})$ *iterations, with probability one the estimated approximation* $\boldsymbol{V}_t$ *satisfies*

$$\|\boldsymbol{V}_t - \boldsymbol{V}^*\|_\infty \leq \frac{3\varepsilon}{1-\gamma}$$

for any choice of initialization $\boldsymbol{V}_0 \in \left[0, \frac{1}{1-\gamma}\right]^{|\mathcal{S}|}$.

Such results provide guidance for choice of parameters. Indeed, the experimental results are consistent with the theory presented above.



(a) SM (b) TM (c) 3-D SM

Figure 1: **Left:** In the standard maze (SM), the player's objective is to navigate to the bottom left. **Middle:** In the terrain maze (TM), the player proceeds to the bottom left corner. Greater costs are incurred for moving uphill than downhill. High positions are indicated by red colors, and low positions are indicated by blue colors. **Right:** An example multidimensional problem. Here too the player's objective is to navigate to the lower-left corner (i.e., $(1, 1, 1)$).

Experiments

To test the theory developed in Sections , we perform a number of numerical experiments.[2] We test our methods on a variety of MDPs of different sizes and complexity. Our results show that state aggregation can achieve faster convergence than standard value iteration; that state aggregation scales appropriately as the size and dimensionality of the underlying MDP increases; and that state aggregation is reasonably robust to measurement error (simulated by adding noise to the action costs) and varying levels of stochasticity in the transition matrix.

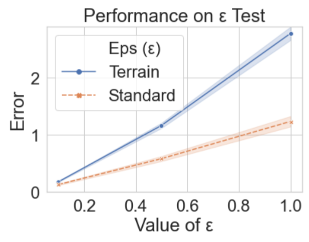
MDP problems

We consider two problems in testing our algorithm. The first, which we term the "standard maze problem" consists of a $d_1 \times d_2 \times \cdots d_n$ grid of positions. Each position is connected to one or more adjacent positions. Moving from position to position incurs a constant cost, except for moving to the terminal state of the maze (the position $(1, \ldots, 1)$) which incurs a constant reward.[3] There is a unique path from each position to the terminal state. A two-dimensional 20×20 standard maze, in which the player can move, depending on their position, up, down left, or right is illustrated in Figure 1a.

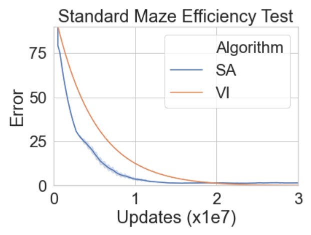
The second problem is the "terrain maze problem." As in a standard maze, each state in the terrain maze represents a position in a $d_1 \times \cdots \times d_n$ grid. As before, we imagine that the player can move from state to state only

[2]All experiments were performed in parallel using forty Xeon E5-2698 v3 @ 2.30GHz CPUs. Total compute time was approximately 60 hours. Code and replication materials are available in the Supplementary Materials.

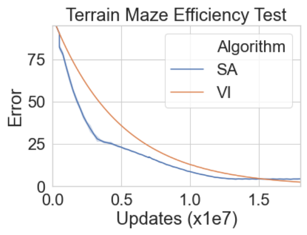
[3]We rescale action costs in both the standard and terrain mazes to ensure that the maximum cost-to-go is exactly 100.



(a) Influence of ε on both mazes



(b) Efficiency test on standard maze



(c) Efficiency test on terrain maze

Figure 2: **Left:** The error of state-aggregated value iteration after convergence as a function of ε. **Middle:** Average convergence speed and 95% confidence intervals of state-aggregated value iteration on 500×500 standard mazes. **Right:** Average convergence speed and 95% confidence intervals of state-aggregated value iteration on 500×500 terrain mazes.

by travelling a single unit in any direction. (The player is only constrained from moving out of the grid entirely.) The player receives a reward for reaching the final square of the maze, which again we place at position $(1, \ldots, 1)$. However, in contrast to the standard maze game, the player's movements incur different costs at different positions. In particular, the maze is determined by a "height function" $H : \{1, \ldots, d_1\} \times \cdots \times \{1, \ldots, d_n\} \to \mathbb{R}$. The cost of movement is set to be the difference in heights between the player's destination position and their current position, normalized appropriately.

In both problems, we also allow for stochasticity controlled by a parameter p which gives the probability that a player moves in their intended direction. For $p = 1$, the MDP is deterministic; otherwise, with probability $1 - p$, the player moves in a different direction chosen uniformly at random.

In both problems, the actions available at any state correspond to a very sparse transition probability vectors, since players are constrained to move along cardinal directions at a rate of a single unit. However, in the standard maze game, the cost-to-go at any position is extremely sensitive to the costs-to-go at states that are very distant. In a 10×10 standard maze, the initial tile (i.e., position (1, 1)) is often between 25 and 30 units away from the destination tile (i.e., position (10, 10)). In contrast, in the terrain maze game, the cost-to-go is much less sensitive to far away positions, because local immediate costs, dictated by the slopes one must climb or go down to move locally, are more significant.

Type	Dims.	Error	95% CI
Trn.	100×100	4.41	± 0.14
Trn.	200×200	4.34	± 0.16
Trn.	300×300	4.65	± 0.18
Trn.	500×500	4.27	± 0.17
Trn.	1000×1000	4.27	± 0.16
Std.	100×100	1.43	± 0.16
Std.	200×200	1.39	± 0.15
Std.	300×300	1.42	± 0.20
Std.	500×500	1.11	± 0.16
Std.	1000×1000	1.40	± 0.16

Type	Dims.	Error	95% CI
Trn.	10^3	1.91	± 0.007
Trn.	10^4	3.02	± 0.009
Trn.	10^5	3.59	± 0.008
Trn.	10^6	3.85	± 0.005
Std.	10^3	1.36	± 0.019
Std.	10^4	1.36	± 0.017
Std.	10^5	1.23	± 0.013
Std.	10^6	1.31	± 0.013

Table 1: Scaling properties of state aggregation value iteration. Reported errors represent the mean value from running each experiment 20 times.

Benchmarks and parameters We measure convergence by the ℓ^∞-distance (hereafter "error") between the current cost-to-go vector and the true cost-to-go vector, and we evaluate the speed based on the size of error and the number of updates performed. Notice that for global value iteration, an update for state s has the form

$$T_s(\boldsymbol{V}) = \min_{a \in \mathcal{A}} \left(r(s,a) + \gamma \boldsymbol{P}_{s,a}^\top \boldsymbol{V} \right),$$

and an update for mega-state j (represented by s) has the form

$$W_t(j) = (1 - \alpha_{t-t_0}) W_{t-1}(j) + \alpha_{t-t_0} T_s \tilde{\boldsymbol{V}}(\boldsymbol{W}_{t-1}).$$

Because the transition matrix is not dense in our examples, the computational resources required for global value iteration update and aggregated update are roughly the same. For each iteration, the value iteration will always perform $|\mathcal{S}|$ updates, and for Algorithm 3, only K updates, one for each mega-state, will be performed if in the aggregation phase.

All experiments are performed with a discount factor $\gamma = 0.95$. We set $|\mathcal{A}_i| = 5$ and $|\mathcal{B}_i| = 2$ for every i, and for learning rate we set $\alpha_t = \frac{1}{\sqrt{t}}$. The cost function is normalized such that $\|\boldsymbol{V}^*\|_\infty = 100$, and we choose aggregation constant to be $\varepsilon = 0.5$ (unless otherwise indicated). We set the initial cost vector $\boldsymbol{V}_0$ to be the zero vector $\mathbf{0}$.

Results

Influence of ε. We test the effect of ε on the error Algorithm 3 produces. We run experiments with aggregation constant $\varepsilon = 0.05, 0.1$, and 0.5 for 500×500 standard and terrain mazes. For each ε, we run 1,000 iterations of Algorithm 3, and each experiment is repeated 20 times; the results, shown in Figure 2a, indicate that the approximation error scales in proportion to ε, which is consistent with Proposition 1 and Theorem 1.

Efficiency. We test the convergence rate of Algorithm 3 against value iteration on 500×500 standard and terrain mazes, repeating each experiment 20 times. From Figure 2b and 2c, we see that state-aggregated value iteration is very efficient at the beginning phase, converging in fewer updates than value iteration.

Scalibility of state aggregation. We run state-abstracted value iteration on standard and terrain mazes of size 100×100, 200×200, 300×300, 500×500, and 1000×1000 for 1000 iterations. We repeat each experiment 20 times, displaying the results in Table 1. Next, we run state-aggregated value iteration on terrain mazes of increasingly large underlying dimension, as shown in the right side of Table 1, likewise for 20 repetitions for each size. The difference with the previous experiment is that not only does the state space increase, the action space also increases exponentially. Our results show that the added complexity of the high-dimensional problems does not appear to substantially affect the convergence of state-aggregated value iteration and our method is able to scale with very large MDP problems.

Type	p	σ	Error	95% CI
Terrain	0.92	0.00	4.44	± 0.24
Terrain	0.92	0.01	4.41	± 0.18
Terrain	0.92	0.05	4.97	± 0.17
Terrain	0.92	0.10	6.36	± 0.19
Terrain	0.95	0.00	4.43	± 0.17
Terrain	0.95	0.01	4.32	± 0.14
Terrain	0.95	0.05	4.93	± 0.17
Terrain	0.95	0.10	6.39	± 0.17
Terrain	0.98	0.00	4.37	± 0.19
Terrain	0.98	0.01	4.31	± 0.14
Terrain	0.98	0.05	5.01	± 0.14
Terrain	0.98	0.10	6.52	± 0.20

Type	p	σ	Error	95% CI
Standard	0.92	0.00	1.39	± 0.19
Standard	0.92	0.01	1.61	± 0.23
Standard	0.92	0.05	2.62	± 0.15
Standard	0.92	0.10	5.56	± 0.46
Standard	0.95	0.00	1.49	± 0.17
Standard	0.95	0.01	1.57	± 0.14
Standard	0.95	0.05	2.86	± 0.17
Standard	0.95	0.10	5.72	± 0.39
Standard	0.98	0.00	1.43	± 0.18
Standard	0.98	0.01	1.88	± 0.16
Standard	0.98	0.05	3.48	± 0.28
Standard	0.98	0.10	5.86	± 0.26

Table 2: Numerical experiments illustrating the robustness of state-aggregated value iteration to stochasticity and noisy action costs. Errors represent the average ℓ_∞-distance to the true cost-to-go values in twenty independent runs.

Robustness. We examine the robustness of state-aggregated value iteration to two sources of noise. We generate 500×500 standard and terrain mazes, varying the level of stochasiticity by setting $p = 0.92, 0.95, 0.98$. We also vary the amount of noise in the cost vector by adding a mean 0, standard deviation $\sigma = 0.0, 0.01, 0.05, 0.1$ normal vector to the action costs. The results, shown in Table 2, indicate that state-aggregated value iteration is reasonably robust to stochasiticity and measurement error.

Continuous Control Problems

We conclude the experiments section by showing the performance of our method on a real-world use case in continuous control. These problems often involve solving complex tasks with high-dimensional sensory input. The idea typically involves teaching an autonomous agent, usually a robot, to successfully complete some task or achieve some goal state. These problems are often very tough as they reside in the continuous state space (and many times action space) domain.

We already showcased the significant reduction in algorithm update costs during learning on grid-world problems in comparison with value iteration. Our goal for this section is to showcase real world examples of how our method may be practically applied in the field of continuous control. This is important not only to emphasize that our idea has a practical use case, but also to further showcase its ability to scale into extremely large (and continuous) dimensional problems.

Environment We choose to use the common baseline control problem in the "CartPole" system. The CartPole system is a typical physics control problem where a pole is attached to a central joint of a cart, which moves along an endless friction-less track. The system is controlled by applying a force to either move to the right or left with the goal to balance the pole, which starts upright. The system terminates if (1) the pole angle is more than 12 degrees from the vertical axis, or (2) the cart position is more than 2.4 cm from the center, or (3) the episode length is greater than 200 timesteps.

During each episode, the agent will receive constant reward for each step it takes. The state-space is the continuous position and angle of the pole for the cartpole pendulum system and the action-space involves two actions - applying a force to move the cart left or right. A more in-depth explanation of the problem can be found in Kumar (2020).

Results Since the CartPole problem is a multi-dimensional continuous control problem, there is no ground truth v-values so we choose to utilize the dense reward nature of this problem and rely on the accumulated reward to quantify algorithm performance as opposed to error. In addition, since value iteration requires discrete states, we discretize all continuous dimensions of the state space into bins and generate policies on the discretized environment. More specifically, we discretize the continuous state space into bins by dividing each dimension of the domain into equidistant intervals.

For our aggregation algorithm, we use $\gamma = 0.99$ following what is commonly used in this problem in past works. We set the initial cost vector V_0 to be the zero vector. We fix $\varepsilon = 0.2$ and set $\alpha_t = \frac{1}{\sqrt{t}}$. We note that given the symmetric nature of this continuous control problem, we do not need to alternate between global and aggregate updates. The adaptive aggregation of states already groups states effectively together for strong performance and significant speedups in the number of updates required. We first do a sweep of number of bins to discretize the problem space to determine the number that best maximizes the performance of value iteration in terms of both update number and reward and found this to be around 2000 bins. We then compare the performances of state aggregation and value iteration on this setting in figure 3. To also further showcase the adaptive advantage of our method, we choose a larger bin (10000) number that likely would have been chosen if no bin sweeping had occurred and show that our method acts as an "automatic bin adjuster" and offers the significant speedups without any prior tuning in figure 3. Interestingly in both situations, the experimental results show that the speedup offered by our method seem significant across different bin settings of the CartPole problem. These results may prove to have strong theoretical directions in future works.

Discussion

Value iteration is an effective tool for solving Markov decision processes but can quickly become infeasible in problems with large state and action spaces. To address this difficulty, we develop adaptive state-aggregated value iteration, a novel method of solving Markov decision processes by aggregating states with similar cost-to-go values and updating

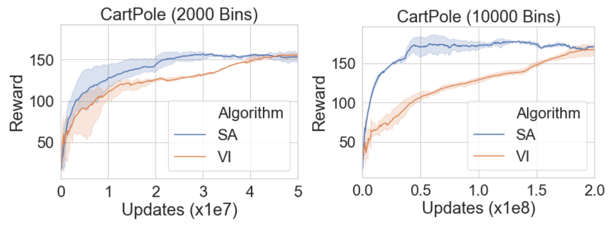


Figure 3: CartPole problem reward versus number of updates comparison between value iteration and our adaptive state aggregation method. Even in use-cases such as continuous control, our method offers valuable speedups in the learning process of MDPs and stays robust against different discretization schemes.

them in tandem. Unlike previous methods of reducing the dimensions of state and action spaces, our method is general and does not require prior knowledge of the true cost-to-go values to form aggregate states. Instead our algorithm learns the cost-to-go values online and uses them to form aggregate states in an adaptive manner. We prove theoretical guarantees for the accuracy, convergence, and convergence rate of our state-aggregated value iteration algorithm, and demonstrate its applicability through a variety of numerical experiments.

State- and action-space reduction techniques are an area of active research. Our contribution in the dynamic state-aggregated value iteration algorithm provides a general framework for approaching large MDPs with strong numerical performances justifying our method. We believe our algorithm can serve as a foundational ground for both future empirical and theoretical work.

We conclude by discussing promising directions for future work on adaptive state aggregation. First, we believe that reducing the number of updates per state by dynamically aggregating states can also be extended more generally in RL settings with model-free methods. Second, our proposed algorithm's complexity is of the same order as value iteration. Future work may seek to eliminate the dependence on γ in the error bound. Lastly, by adaptively choosing ε, it may be possible to achieve better complexity bounds not only for planning problems but also for generative MDP models (Sidford et al. 2018b,a).

References

Abel, D.; Arumugam, D.; Asadi, K.; Jinnai, Y.; Littman, M. L.; and Wong, L. L. 2019. State abstraction as compression in apprenticeship learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 3134–3142.

Abel, D.; Hershkowitz, D.; and Littman, M. 2016. Near optimal behavior via approximate state abstraction. In *International Conference on Machine Learning*, 2915–2923. PMLR.

Abel, D.; Umbanhowar, N.; Khetarpal, K.; Arumugam, D.; Precup, D.; and Littman, M. 2020. Value preserving state-action abstractions. In *International Conference on Artificial Intelligence and Statistics*, 1639–1650. PMLR.

Anderson, D. G. 1965. Iterative procedures for nonlinear integral equations. *Journal of the ACM (JACM)*, 12(4): 547–560.

Azar, M. G.; Munos, R.; and Kappen, H. J. 2013. Minimax PAC bounds on the sample complexity of reinforcement learning with a generative model. *Machine learning*, 91(3): 325–349.

Baras, J.; and Borkar, V. 2000. A learning algorithm for Markov decision processes with adaptive state aggregation. In *Proceedings of the 39th IEEE Conference on Decision and Control (Cat. No. 00CH37187)*, volume 4, 3351–3356. IEEE.

Bellman, R. 1957. A Markovian Decision Process. *Indiana Univ. Math. J.*, 6: 679–684.

Bertsekas, D. P. 2018. Feature-based aggregation and deep reinforcement learning: A survey and some new implementations. *IEEE/CAA Journal of Automatica Sinica*, 6(1): 1–31.

Bertsekas, D. P.; Castanon, D. A.; et al. 1988. Adaptive aggregation methods for infinite horizon dynamic programming.

Dean, T.; Givan, R.; and Leach, S. 1997. Model Reduction Techniques for Computing Approximately Optimal Solutions for Markov Decision Processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, UAI'97, 124–131. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 1558604855.

Duan, Y.; Ke, Z. T.; and Wang, M. 2018. State aggregation learning from markov transition data. *arXiv preprint arXiv:1811.02619*.

Fang, H.-r.; and Saad, Y. 2009. Two classes of multisecant methods for nonlinear acceleration. *Numerical Linear Algebra with Applications*, 16(3): 197–221.

Ferns, N.; Panangaden, P.; and Precup, D. 2012. Metrics for Markov decision processes with infinite state spaces. *arXiv preprint arXiv:1207.1386*.

Herzberg, M.; and Yechiali, U. 1994. Accelerating procedures of the value iteration algorithm for discounted Markov decision processes, based on a one-step lookahead analysis. *Operations Research*, 42(5): 940–946.

Hostetler, J.; Fern, A.; and Dietterich, T. 2014. State aggregation in Monte Carlo tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28.

Jiang, N.; Singh, S.; and Lewis, R. 2014. Improving UCT planning via approximate homomorphisms. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, 1289–1296.

Jong, N. K.; and Stone, P. 2005. State Abstraction Discovery from Irrelevant State Variables. In *IJCAI*, volume 8, 752–757. Citeseer.

Kumar, S. 2020. Balancing a CartPole System with Reinforcement Learning - A Tutorial. *CoRR*, abs/2006.04938.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a Unified Theory of State Abstraction for MDPs. *ISAIM*, 4: 5.

McCallum, R. 1997. Reinforcement learning with selective perception and hidden state.

Mendelssohn, R. 1982. An iterative aggregation procedure for Markov decision processes. *Operations Research*, 30(1): 62–73.

Ortner, R. 2013. Adaptive aggregation for reinforcement learning in average reward Markov decision processes. *Annals of Operations Research*, 208(1): 321–336.

Robbins, H.; and Monro, S. 1951. A stochastic approximation method. *The annals of mathematical statistics*, 400–407.

Schweitzer, P. J.; Puterman, M. L.; and Kindle, K. W. 1985. Iterative aggregation-disaggregation procedures for discounted semi-Markov reward processes. *Operations Research*, 33(3): 589–605.

Shlakhter, O.; Lee, C.-G.; Khmelev, D.; and Jaber, N. 2010. Acceleration operators in the value iteration algorithms for Markov decision processes. *Operations research*, 58(1): 193–202.

Sidford, A.; Wang, M.; Wu, X.; Yang, L. F.; and Ye, Y. 2018a. Near-optimal time and sample complexities for solving Markov decision processes with a generative model. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 5192–5202.

Sidford, A.; Wang, M.; Wu, X.; and Ye, Y. 2018b. Variance reduced value iteration and faster algorithms for solving markov decision processes. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, 770–787. SIAM.

Sinclair, S. R.; Banerjee, S.; and Yu, C. L. 2019. Adaptive discretization for episodic reinforcement learning in metric spaces. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(3): 1–44.

Slivkins, A. 2011. Contextual bandits with similarity information. In *Proceedings of the 24th annual Conference On Learning Theory*, 679–702. JMLR Workshop and Conference Proceedings.

Tsitsiklis, J. N.; and Van Roy, B. 1996. Feature-based methods for large scale dynamic programming. *Machine Learning*, 22(1): 59–94.

Van Roy, B. 2006. Performance loss bounds for approximate value iteration with state aggregation. *Mathematics of Operations Research*, 31(2): 234–244.

Wasan, M. T. 2004. *Stochastic approximation*. 58. Cambridge University Press.

Zhang, J.; O'Donoghue, B.; and Boyd, S. 2020. Globally convergent type-I Anderson acceleration for nonsmooth fixed-point iterations. *SIAM Journal on Optimization*, 30(4): 3170–3197.