Minimal Residual Approaches for Policy Evaluation in Large Sparse Markov Chains

Yao Hengshuai Liu Zhiqiang
School of Creative Media, City University of Hong Kong
Tat Chee Avenue, Kowloon, Hong Kong, S.A.R., P.R. CHINA
hengshuai@gmail.com

Abstract

We consider the problem of policy evaluation in a special class of Markov Decision Processes (MDPs) where the underlying Markov chains are *large* and *sparse*. We start from a stationary model equation that the limit of Temporal Difference (TD) learning satisfies, and develop a Robbins-Monro method consistently estimating its coefficients. Then we introduce the minimal residual approaches, which solve an approximate version of the stationary model equation. Incremental Least-squares temporal difference (iLSTD) is shown to be a special form of minimal residual approaches. We also develop a new algorithm called minimal residual (MR) algorithm whose step-size can be computed on line. We introduce the Compressed Sparse Row (CSR) format and reduce the complexity of MR to near that of TD. The advantages of the MR algorithm are that it has comparable data efficiency and computational efficiency to iLSTD, but does not require manual selection of step-size.

1 Introduction

In the practice of reinforcement learning (RL), the problems often have a large state space. If the state is represented by the lookup table, the dimension of the feature vector equals the number of states. For problems with a large state space, this will cause what is called *Bellman's curse of dimensionality* (Bertsekas & Tsitsiklis 1996). To tackle this problem, people use function approximation (Bertsekas & Tsitsiklis 1996; Sutton & Barto 1998). However, for complex tasks, the number of features is still not too small, which leads to a high-dimensional feature vector.

In this paper, we consider a special class of RL tasks, where transitions among states are *sparse*, in the sense that each predecessor has only a few successors. Many RL tasks were found to have this feature (Moore & Atkeson 1993). Examples are Sutton's random walks (Sutton 1988), Boyan chain example (Boyan 1999;2002), and some continuous tasks solved by discretization such as cart-pole balancing task, *etc.*. For simplicity, we focus only on the Markov Decision Processes (MDPs) in this paper. Extensions to general reinforcement learning can be easily made following similar procedures in (Lagoudakis & Parr 2002).

We assume that each state is visited infinitely often as long as the simulation stage of the MDPs is infinitely long. Another assumption is that the linear function approximation used does not vary the sparse nature of the Markov chain, *i.e.*, the feature of each state is also sparse. While this seems a restrictive condition, several popular linear function approximation schemes such as Sutton's lookup table (Sutton 1988), Boyan's linear interpolation approximation (Boyan 1999;2002), and tile coding (Sutton & Barto 1998) are indeed sparse (Geramifard, Bowling, & Sutton 2006).

We are motivated by recent work on data efficiency and computational efficiency of RL algorithms. Boyan proposed Least-squares temporal difference (LSTD) learning to improve the data efficiency of temporal difference (TD) learning, but the computation cost of LSTD is quite high, which generally requires $O(K^3)$ per time step, where K is the number of features (Boyan 1999;2002). Recursive LSTD (RLSTD) iteratively computes the inversion of some matrix and reduces the complexity to $O(K^2)$ (Bradtke & Barto 1996). A remarkable merit of LSTD is that it has no parameter to tune by hand. This feature makes the algorithm quite convenient for users, and very useful when the domains are complex.

Incremental LSTD (iLSTD) is a new algorithm (Geramifard, Bowling, & Sutton 2006), recently proposed to strive a balance between LSTD and TD. Its predictions are almost as accurate as LSTD, but its computational cost is considerably lower. For sparse RL tasks, the complexity of iLSTD is only slightly higher than TD. They showed that iLSTD can be more efficient by updating only a small portion of states with the largest TD updates. This paper inducts the role of a key vector used by iLSTD, and proves that iLSTD is a special minimal residual approach. iLSTD also requires tuning the step-size by hand, which involves experimentally finding two constant parameters.

This paper has the same research target with iLSTD, but is built on a more general setting, and aims to develop an algorithm that alleviates the labor of parameter tuning as LSTD does. We introduce a class of minimal residual approaches that include iLSTD and a new algorithm, which we call minimal residual (MR) algorithm. The step-size of MR can be computed on line and does not require selecting manually. Moreover, we also develop a new method to reduce the complexity of MR to near that of TD. Our method is built on the compressed storage scheme of sparse matrix, and seems to be applicable to other RL algorithms. Specifically, we use

the Compressed Sparse Row (CSR) format to store a sparse matrix involved in MR. Based on CSR, the complexity of MR per time step is reduced to $O(qK)$, where q is a small positive real related to the sparsity of the matrix used by MR.

In the remaining of this section, we define a stationary model equation and review recent results on it. In Section 2, we define the stochastic model equation and use a Robbins-Monro (RM) procedure to estimate its coefficients on line. We also prove the consistency of the RM procedure in this section. In Section 3, we introduce the general minimal residual approaches, which solve the stochastic model equation according to the latest residual. It is shown that iLSTD is a special form of the minimal residual approaches. We continue with developing a minimal residual (MR) algorithm that uses a new rule of step-size. In Section 4, we introduce the CSR format and the efficient implementation of MR. In Section 5, we compare MR algorithm with TD/LSTD/iLSTD on Boyan chain example. Results show that MR achieves similar performances with iLSTD: it has comparable data efficiency with LSTD, but its computational complexity is only slightly higher than TD. Finally in Section 6, we discuss some future directions.

1.1 Stationary model equation

Assume we have a state space $\mathcal{S} = \{1, 2, \ldots, N\}$. The problem of *policy evaluation*, or *learning to predict*, is formalized as predicting the long-term optimal reward for each state as the agent interacts with the environment. In particular, the long-term optimal reward for state s is

$$J(s) = \sum_{k=0}^{\infty} \gamma^k r(s_k, s_{k+1}), \quad s_0 = s,$$

where $r(s_k, s_{k+1})$ is the reward received by the agent at time k, and $\gamma \in (0, 1]$ is the discount factor.

We consider a linear function approximation. Given $K (K \leq N)$ feature functions,

$$\varphi_k(\cdot) : \mathcal{S} \mapsto \mathcal{R}^K, \quad k = 1, \ldots, K,$$

we define the following feature vector:

$$\phi(s_i) = [\varphi_1(s_i), \varphi_2(s_i), \ldots, \varphi_K(s_i)]'.$$

For simplicity, we will denote $\phi(s_i)$ as ϕ_i. The long-term optimal reward vector can now be approximated by $\hat{J} = \Phi w$, where w is the weight vector to be learned, and

$$\Phi'(k, j) = \varphi_k(j), \quad k = 1, \ldots, K; \quad j = 1, \ldots, N.$$

For an ergodic Markov chain that has steady-state probabilities $\pi(1), \pi(2), \ldots, \pi(N)$, Tsitsiklis and Van Roy (1997) proved that TD finally finds a weight vector w^* that satisfies a linear system of equations:

$$Aw^* + b = 0, \tag{1}$$

where A and b are defined by

$$A = \Phi' D(\gamma P - I) \sum_{k=0}^{\infty} (\lambda\gamma P)^k \Phi,$$

and

$$b = \Phi' D \sum_{k=0}^{\infty} (\lambda\gamma P)^k \bar{r},$$

D is the diagonal matrix with diagonal entries $\pi(i)$, $i = 1, \ldots, N$; $\lambda \in [0, 1]$ is the eligibility trace factor; P is the transition probability matrix of the Markov chain; I is the identity matrix; and $\bar{r}$ is the vector with components $\bar{r}_i = \sum_{j=1}^{N} P_{i,j} r(i, j)$, $i = 1, \ldots, N$. For each $\lambda \in [0, 1]$, w^* is also the limit point of LSTD(λ) and iLSTD(λ) .

However, the difficulty with reinforcement learning is that the coefficients of (1) are not explicitly known, due to the unavailability of the model in terms of P and $\bar{r}$. Thus equation (1) only serves for analysis and not applicable in practice. We will call (1) the stationary model equation.

1.2 Law of Large Numbers

The basic idea of LSTD is to accumulate the experience as the agent explores in the environment. This process grows some data structure that stores the experience. If the current transitioning from s_t to s_{t+1} incurs a reward r_t, a matrix and a vector are updated by

$$\begin{cases} \tilde{A}_{t+1} = \tilde{A}_t + \Delta\tilde{A}_t \\ \tilde{b}_{t+1} = \tilde{b}_t + \Delta\tilde{b}_t, \end{cases} \tag{2}$$

where

$$\Delta\tilde{A}_t = z_t(\gamma\phi(s_{t+1}) - \phi(s_t))', \tag{3}$$

$$\Delta\tilde{b}_t = z_t r_t, \tag{4}$$

z_t is the eligibility trace updated by

$$z_{t+1} = \lambda z_t + \phi_{t+1}.$$

Because the components of $\tilde{A}_{t+1}$ and $\tilde{b}_{t+1}$ can get to infinity, and not eligible for theoretical analysis, it is better to use some well-define term. For infinite horizon problems simulated by a single trajectory that is long enough, Tadić (2001) and Nedić & Bertsekas (2003) used the following normalized data structure,

$$\begin{cases} A_{t+1} = \frac{1}{t+1}\tilde{A}_{t+1} \\ b_{t+1} = \frac{1}{t+1}\tilde{b}_{t+1}. \end{cases} \tag{5}$$

They independently proved that (5) satisfies the law of large number. To be precise, under certain conditions, the two estimations given by (5) converge with probability one to A and b, respectively. Their ideas provide a starting point for this paper, but the limitation of (5) is that the two estimations can not be applied to finite horizon problems such as Sutton's random walks (Sutton 1988) and Boyan chain example (Boyan 1999;2002).

2 Stochastic model equation

In this section, we extend (5) to finite horizon problem. Typically the finite horizon problem here is generated by absorbing MDPs. For simplicity, we focus only on $\lambda = 0$. Given an initial state s_0, each simulation stage of the absorbing MDPs generates a finite trajectory $s_0, \ldots, s_q$, where s_q is the absorbing state. Assume we have M trajectories and the

mth trajectory is of length L_m. In this case, a consistent estimation of A would be

$$A_M = \frac{1}{T}\sum_{m=1}^{M}\sum_{k=0}^{L_m}\phi_k(\gamma\phi(s_{k+1}) - \phi(s_k))', \qquad (6)$$

where T is the number of all observed state visits in M trajectories. Similarly, we define

$$b_M = \frac{1}{T}\sum_{m=1}^{M}\sum_{k=0}^{L_m}\phi_k r_k. \qquad (7)$$

We then have the following lemma and theorem.

Lemma 1. *Assume the Markov chain is ergodic. Also assume that in all M trajectories, $f_i(T)$ is the number of times that state i is visited; $f_{ij}(T)$ is the number of times that state i transitions to state j . We have*

$$\frac{f_i(T)}{T} \to \pi(i), \quad T \to \infty, \quad w.p.1. \quad \forall i \in \mathcal{S}.$$

$$\frac{f_{ij}(T)}{f_i(T)} \to P_{ij}, \quad T \to \infty, \quad w.p.1. \quad \forall i, j \in \mathcal{S}.$$

Proof. The essence of this lemma is based on the law of large numbers. Similar arguments can be made according to those in Lemma 4.2(a) by Nedić and Bertsekas (2003). □

Theorem 1. *Assume matrix Φ has full column rank. For ergodic Markov chains, the following relations hold:*

(a). Let D_M and $A_M^{(1)}$ be defined as follows:

$$D_M = \frac{1}{T}\sum_{m=1}^{M}\sum_{k=0}^{L_m}\phi_k\phi_k', \qquad (8)$$

$$A_M^{(1)} = \frac{1}{T}\sum_{m=1}^{M}\sum_{k=0}^{L_m}\phi_k\phi_{k+1}'. \qquad (9)$$

Then as $M \to \infty$, we have

$$D_M \to \Phi' D \Phi, \quad w.p.1,$$

$$A_M^{(1)} \to \Phi' D P \Phi, \quad w.p.1.$$

(b). $A_M \to A, w.p.1, \quad$ and $\quad b_M \to b, \quad M \to \infty, \quad$ w.p.1.

Proof. Proof of (a). We rewrite D_M as

$$\frac{1}{T}\sum_{m=1}^{M}\sum_{k=0}^{L_m}\phi_k\phi_k' = \frac{1}{T}\sum_{m=1}^{M}\sum_{k=0}^{L_m}\phi_k \cdot 1 \cdot \phi_k'.$$

Observing that the scalar 1 can be written as

$$1 = [0, \quad \ldots, \quad \underset{s_k}{1}, \quad \ldots, \quad 0][0, \ldots, \quad \underset{s_k}{1}, \quad \ldots, 0]',$$

where the unit vector is of length N. Therefore, we have

$$\frac{1}{T}\sum_{m=1}^{M}\sum_{k=0}^{L_m}\phi_k\phi_k' = \Phi'\left[\frac{f_{s_k}(T)}{T}\right]_{s_k,s_k}\Phi,$$

where $[x]_{i,i}$ stands for a diagonal matrix whose (i,i)th entry is x. According to Lemma 1, we arrive that $D_{t+1} \to \Phi' D \Phi$, *w.p.1.*

Next we prove (9).

$$\begin{aligned} A_M^{(1)} &= \Phi'\left[\frac{f_{s_k,s_{k+1}}(T)}{T}\right]_{s_k,s_{k+1}}\Phi \\ &= \Phi'\left[\frac{f_{s_k,s_{k+1}}(T)}{f_{s_k}(T)} \cdot \frac{f_{s_k}(T)}{T}\right]_{s_k,s_{k+1}}\Phi \\ &\overset{Lemma1}{\longrightarrow} \Phi' D P \Phi. \end{aligned}$$

Proof of (b). Here we only prove for A_M. The analysis of b_M is similar. The proof of (b) is direct from (a) by observing that $A_M = -D_M + \gamma A_M^{(1)}$. □

Estimations (6) and (7) can also be updated in an iterative manner immediately after a transition, which is a Robbins-Monro procedure. After a transition from s_t to s_{t+1}, we update our estimations by

$$A_{t+1} = A_t + \frac{1}{T}(\phi_t(\gamma\phi_{t+1} - \phi_t)' - A_t), \qquad (10)$$

and

$$b_{t+1} = b_t + \frac{1}{T}(\phi_t r_t - b_t), \qquad (11)$$

where T is the total number of state visits. After (10) and (11), T is updated by $T = T + 1$. Note that the subscript of A and b here becomes the transition index. Without special note, we will use this convention in the remaining of this paper.

Theorem 1 tells that although the stationary model equation is not explicitly known or of practical use, we can get an approximation of it by estimating its coefficients. In the following we define an approximate version of the stationary model equation that is of practical use, which provides a ground for our later approaches.

Definition 1 (Stochastic model equation)**.** *Given A_{t+1} and b_{t+1} defined by (10) and (11), we call the equation*

$$A_{t+1}w + b_{t+1} = 0,$$

the stochastic model equation*.*

Another motivation for defining the stochastic model equation is that it relates the learning of the weight vector to the accumulated experience contained in the data structure (A_{t+1}, b_{t+1}). By the stochastic model equation, the experience of agent is passed to the weight vector. In this way, experience is finally availed for prediction. As A_{t+1} and b_{t+1} become more accurate, the solution of the stochastic model equation will also solve the stationary model equation in more accuracy. Thus we learn the optimal weights by solving the stochastic model equation. To do this, we introduce the general minimal residual approaches that minimize some residual error at each time step.

3 Minimal Residual Approaches

Given the current estimations of A, b and w, we have a residual vector,

$$\delta_t = A_{t+1}w_t + b_{t+1}. \tag{12}$$

This residual stands for the difference between the two sides of the stochastic model equation. The general minimal residual approach is cast as

$$w_{t+1} = w_t + \alpha_t \delta_t, \tag{13}$$

where α_t is some step-size. Because A_{t+1} converges to the negative definite matrix A with probability one, the step-size is required to be positive in order to guarantee the convergence of the algorithm.

At time t, the general minimal residual approach minimizes the *residual error* $||\delta_t||_2^2$. Therefore, the difference between the two sides of the stochastic model equation is gradually reduced. As this difference is progressively approaching that between the stationary model equation, the weight vector learned by minimal residual approach will solve the stationary model equation in the long run.

3.1 iLSTD as a minimal residual approach

We show that iLSTD is a special minimal residual approach, of the form given by (12) and (13).

The original induction of iLSTD is given in (Geramifard, Bowling, and Sutton 2006). They also have an advanced version with higher computational efficiency, which will be considered in Section 4. At time t, after steps (2), (3) and (4), a vector μ is updated for the first time:

$$\mu_t(w_t) = \mu_{t-1}(w_t) + \Delta\tilde{A}_t w_t + \Delta\tilde{b}_t. \tag{14}$$

And the update of the weight vector follows after the first update of μ:

$$\Delta w_t = \bar{\alpha}_t \mu_t,$$

where $\bar{\alpha}_t$ is some diminishing step-size.

After the weight update, vector μ is updated for the second time:

$$\mu_t(w_{t+1}) = \mu_t(w_t) + \tilde{A}_{t+1}\Delta w_t. \tag{15}$$

It is not clear what the function of μ is by this induction. Here we induct that $\mu(w_t)/T$ is an estimation of $A_{t+1}w_t + b_{t+1}$.

First let us define $\mu_k = \mu_k(w_k)$. According to equations (14) and (15), we have

$$\begin{aligned}\mu_{k+1} &= \mu_k(w_{k+1}) + \Delta\tilde{A}_{k+1}w_{k+1} + \Delta\tilde{b}_{k+1} \\ &= \mu_k + \tilde{A}_{k+1}\Delta w_k + \Delta\tilde{A}_{k+1}w_{k+1} + \Delta\tilde{b}_{k+1}.\end{aligned}$$

Writing Δw_k as $w_{k+1} - w_k$, according to (2), (3) and (4), we have

$$\begin{aligned}\mu_{k+1} &= \mu_k + \tilde{A}_{k+1}(w_{k+1} - w_k) + \Delta\tilde{A}_{k+1}w_{k+1} + \Delta\tilde{b}_{k+1} \\ &= \mu_k + (\tilde{A}_{k+1} + \Delta\tilde{A}_{k+1})w_{k+1} - \tilde{A}_{k+1}w_k + \Delta\tilde{b}_{k+1} \\ &= \mu_k + \tilde{A}_{k+2}w_{k+1} - \tilde{A}_{k+1}w_k + \Delta\tilde{b}_{k+1} \\ &= \mu_k + (\tilde{A}_{k+2}w_{k+1} + \tilde{b}_{k+2}) - (\tilde{A}_{k+1}w_k + \tilde{b}_{k+1}).\end{aligned}$$

Thus we get the relation of μ_k and μ_{k-1}, described by:

$$\mu_k = \mu_{k-1} + (\tilde{A}_{k+1}w_k + \tilde{b}_{k+1}) - (\tilde{A}_k w_{k-1} + \tilde{b}_k). \tag{16}$$

Summing both sides of equation (16) for $k = 1, \ldots, t$, we have

$$\mu_t = \mu_0 + \tilde{A}_{t+1}w_t + \tilde{b}_{t+1} - (A_0 w_0 + b_0). \tag{17}$$

Dividing both sides of equation (17) by T, producing:

$$\frac{\mu_t}{T} = \frac{\mu_0 - (A_0 w_0 + b_0) + \tilde{A}_{t+1}w_t + \tilde{b}_{t+1}}{T}.$$

According to Theorem 1, for large T, we have

$$\frac{\mu_t}{T} \approx \frac{\tilde{A}_{t+1}w_t + \tilde{b}_{t+1}}{T} = A_{t+1}w_t + b_{t+1},$$

since the initial settings of μ_0, w_0, A_0 and b_0 are all finite.

In (Boyan 1999;2002) and (Geramifard, Bowling, & Sutton 2006), the diminishing step-size is of the following form:

$$\bar{\alpha}_t = \frac{c_0 c_1}{i + c_1}, \tag{18}$$

where i is the trajectory index. Thus iLSTD using (18) is a special case of (13), whose step-size is

$$\alpha_t = \frac{c_0 c_1 T}{i + c_1}. \tag{19}$$

This step-size has two parameters to be selected by hand, and cannot be tuned on line, usually chosen by experimental finding.

We have in fact proved the following theorem.

Theorem 2. *iLSTD using step-size (18) is a variant of the general minimal residual approach given by (12) and (13), taking the step-size from (19).*

3.2 Minimal Residual (MR) algorithm

At time t, we actually get two residual vectors. The residual δ_t is an "old" one, because it is obtained before the weight update. After the weight update, another new residual vector

$$\theta_{t+1} = A_{t+1}w_{t+1} + b_{t+1},$$

is obtained using the latest weight vector. Because θ_{t+1} stands for an improved difference between the two sides of the stochastic model equation, naturally we hope that the new residual error is smaller than the old one.

The new residual vector can be expressed in terms of the old residual, as shown by the following induction:

$$\begin{aligned}\theta_{t+1} &= A_{t+1}(w_t + \alpha_t\delta_t) + b_{t+1} \\ &= \delta_t + \alpha_t A_{t+1}\delta_t,\end{aligned}$$

where the first relation is according to equation (13). Thus the three vectors: δ_t, θ_{t+1}, and $\alpha_t A_{t+1}\delta_t$ form a triangle.

The derivation of the step-size is to require that θ_{t+1} be orthogonal to $\alpha_t A_{t+1}\delta_t$. Thus we are assured that the new residual error is strictly smaller than the old one, *i.e.*,

$$||\theta_{t+1}||_2^2 < ||\delta_t||_2^2.$$

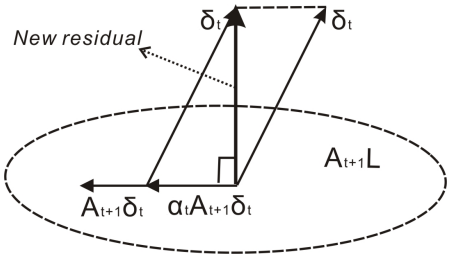

Figure 1: Derivation of the step-size and the projection process.

Therefore, we require that

$$(\alpha_t A_{t+1}\delta_t)'\theta_{t+1} = 0,$$

which gives a new rule of step-size,

$$\alpha_t = -\frac{\delta_t' A_{t+1}\delta_t}{(A_{t+1}\delta_t)'(A_{t+1}\delta_t)}. \tag{20}$$

It is interesting to note that step-size (20) is positive in the long run, because matrix A_{t+1} is negative definite with probability one. This step-size is the optimal value of an optimization problem that minimizes the new residual error over $\alpha \in \mathcal{R}$, *i.e.*,

$$\alpha_t = \min_{\alpha} ||\theta_{t+1}||^2.$$

We call the minimal approach using step-size (20) the *minimal residual* (MR) algorithm. At time t, the computing order of MR can be specified as: (10)→(11)→(12)→(20)→(13). We will call this type of MR the *ordinary* MR algorithm, which contrasts to the compressed implementation of MR algorithm in Section 4.

Assume δ_t is in the space L. By the adaptive step-size (20), $-\alpha_t A_{t+1}\delta_t$ is an orthogonal projection of δ_t onto the space $A_{t+1}L$, as shown by Figure 1.

4 Efficient implementation

Note that steps (10), (12) and (20) are the most complex steps in ordinary MR, which generally require $O(K^2)$ complexity. However, if RL tasks have a sparse nature, the complexity can be reduced. The key observation of (12) and (20) is that the two steps are all matrix-vector multiplication, and only nonzero entries of A_{t+1} contribute to the weight vector. Another perspective is that the zero entries of A_{t+1} imply that "no experience is available for the states related to these entries". Therefore, there is no need to store these void experience. We only have to store the valid experience contained in A_{t+1}. Here we use the Compressed Sparse Row format, which is widely used in the computation of sparse matrix (Saad 2003). In addition, we use some algebraic tricks to avoid explicit computation of (10) in this section.

4.1 Compressed Sparse Row (CSR) format

Assume l_t is the number of nonzero entries in A_t. The Compressed Sparse Row (CSR) format $\mathbb{S}_A^t$, is composed of the following data structure:

- a real array a_t of length l_t, containing all the real values of the nonzero elements of A_t, stored row by row;
- an integer array c_t of length l_t, containing the column indices of the elements stored in a_t;
- a second integer array d_t of length $K+1$, containing the pointers to the beginning of each row in a_t and c_t. The last element $d_t(K+1)$ is always set to $l_t + 1$.

Algorithm 1: Efficient computation of matrix-vector multiplication using CSR.

Data: $\mathbb{S}_A^{t+1}$ and a vector β_t
Result: A vector $o_t = A_{t+1}\beta_t$.
for $k = 1, 2, \ldots, K$ **do**
  $k_1 = d_{t+1}(k)$;
  $k_2 = d_{t+1}(k+1) - 1$;
  $o_t(k) = a_{t+1}(k_1 : k_2)'\beta_t(c_{t+1}(k_1 : k_2))$.
endfor

Algorithm 2: Efficient MR ($\lambda = 0$) using CSR.

Data: A Markov chain and a reward scheme that can be both sampled on line; a feature scheme Φ.
Result: A weight vector w^*, such that $||\Phi w^* - J||_2$ is minimized.
Initialize $\mathbb{S}_A^0$, b_0 and w_0 such that $A_0\delta_0 \neq 0$;
Set $T = 0$;
for *trajectories* $m = 1, \ldots, M$ **do**
  Choose a start state $s_0 \in \mathcal{S}$;
  Set $z_0 = \phi_0$;
  Set $T = T + 1$;
  while s_t *not at end of this trajectory* **do**
    Transition to the next state s_{t+1} and receive a reward r_t;
    Compute y_t using Algorithm 1;
    Update $\mathbb{S}_A^t$; /*According to (10) */
    Update b_{t+1} according to (11);
    Compute x_{t+1} by (21);
    Compute the step-size α_t by equation (20);
      /*equation (20) makes a call to Algorithm 1*/
    Update w_{t+1} according to (22);
    Set $T = T + 1$.
  endw
endfor

4.2 Efficient MR Using CSR

Using CSR, it is very convenient to locate all nonzero entries of A_t, which are stored in array a_t. The index of the first nonzero entry in the kth row of A_t is given by $d_t(k)$, and the index of the last nonzero entry in the kth row is given by $d_t(k+1) - 1$. Thus the nonzero entries in the ith row are $a_t(d_t(k)), a_t(d_t(k)+1), \ldots, a_t(d_t(k+1)-1)$, which we will denote as $a_t(d_t(k) : d_t(k+1) - 1)$ for simplicity.

Now we can efficiently compute the matrix-vector multiplication for sparse A_{t+1} using CSR, which only considers its nonzero entries. The operation process is shown by Algorithm 1.

The computation of residual vector (12) and step-size (20) can both avail Algorithm 1, which takes $\beta_t = w_t$ and $\beta_t =$

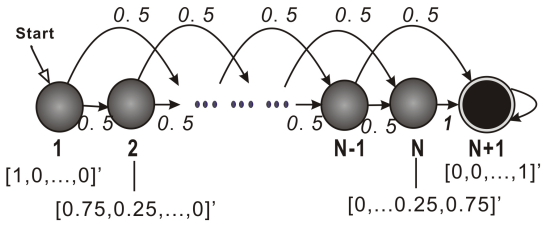

Figure 2: A finite horizon problem: Boyan chain example with $N+1$ states. The transition probability is marked on the arch.

δ_t respectively. The complexity of Algorithm 1 is given in the following lemma.

Lemma 2 (Complexity of Algorithm 1)**.** *The complexity of algorithm 1 is $O(l_{t+1})$.*

When A_{t+1} is sparse, we do not have to update it in the manner of (10). First we define

$$\begin{cases} x_{t+1} = A_{t+1} w_t \\ y_t = A_t w_t, \end{cases}$$

Obviously y_t can be computed by Algorithm 1. The computation of x_{t+1} is direct from y_t. Multiplying both sides of (10) with w_t, we have

$$x_{t+1} = y_t + \frac{1}{T} \{\phi_t [(\gamma \phi_{t+1} - \phi_t)' w_t] - y_t\}. \qquad (21)$$

The weight update can now be specified as

$$w_{t+1} = w_t + \alpha_t (x_{t+1} + b_{t+1}). \qquad (22)$$

The details of MR using CSR are specified by Algorithm 2.

Let us examine the complexity of Algorithm 2.

Theorem 3 (Complexity of Algorithm 2)**.** *The per-time-step complexity of MR using CSR is $O(qK)$, where q is a small positive real related to the sparsity of matrix A.*

Proof. The steps computing y_t and α_t both use Algorithm 1. According to Lemma 2, their complexities are $O(l_t)$. The computation of b_{t+1}, x_{t+1} and w_{t+1} are all $O(K)$.

Assume after t_0 steps, no new nonzero entry of A_{t+1} emerges. In the first t_0 steps, the update of CSR have to insert new entries into a_t, and shift the remaining entries forward. Accordingly, the other two arrays in $\mathbb{S}_A^t$ should also be updated. The total complexities of these operations are $O(l_t)$. After t_0 steps, we only have to update the values of a_t, and do not need to update the other two arrays in $\mathbb{S}_A^t$. Thus the complexity of CSR update is still $O(l_t)$. Assume there are l nonzero entries in A. Obviously l_t converges to l with probability one. As a widely accepted criterion, usually a sparse matrix has only $l < qK$ nonzero entries, where q is some small positive real. Therefore, the complexity of Algorithm 2 is $O(l) < O(qK)$. □

5 Simulation

Boyan chain and the function approximation are shown in Figure 2. Transition from N to $N+1$ incurs a reward -2; transition from $N+1$ incurs 0; the other transitions incur -3. The discount factor γ is set to 1. Under Boyan's linear function approximation, the feature of state $4i-3$ ($i = 1, \ldots, N/4+1$) is set to the unit vector with a single one in the ith row. The features of the other states are the linear interpolation between these unit vectors. Thus we need $K = N/4 + 1$ features. The kth component of w^* is $-8(K-k), k = 1, \ldots, K$.

There is another version of iLSTD (Geramifard, Bowling, & Sutton 2006). The version takes advantage of the sparsity of Δw. It is more computationally efficient than the ordinary iLSTD discussed in Section 3: at each time step, only m components in vector μ (defined in equation (15)) with the largest magnitude are updated. Accordingly, the corresponding m components of w have a priority to be updated. This efficient version was also compared, which is denoted as $m-$iLSTD for ease.

The step-size of iLSTD/$m-$iLSTD is scheduled as follows: c_0 was chosen from the set $\{0.1, 0.01, 0.001, 0.0001\}$, and c_1 was chosen from $\{100, 1000, 10000, 100000\}$. We tried all the sixteen combinations of c_0 and c_1. The performance of iLSTD/$m-$iLSTD in Figure 3 used the best one of them, which selected $c_0 = 0.001$ and $c_1 = 1000$ by experimental finding. For iLSTD, $m-$iLSTD and MR, A_0 was set to $-0.01I$; w_0 and b_0 were set to $[0.1, \ldots, 0.1]'_{K\times 1}$. For LSTD, $\tilde{A}_0$ was set to $-0.01I$ and $\tilde{b}_0$ was set to $[0.1, \ldots, 0.1]'_{K\times 1}$. All compared algorithms used $\lambda = 0$. The priority number m was set to 1 in all figures presented.

In Figure 3, we plot the $2-$norm modeling error of the RM procedure (equation (10) and (11)). It is shown that the RM estimations are becoming more accurate as more trajectory data is experienced. We also plot the RMS errors of LSTD, iLSTD, $m-$iLSTD and MR in Figure 3. All algorithms used the same data set and all RMS errors were averaged over 10 independent runs. An interesting observation is that the RM estimations are much more accurate than the predictions made by any algorithm. This indicates that the accuracy of learning is based on the trajectory data that expresses the inherent model.

As shown in Figure 3, the prediction of iLSTD is more accurate than $m-$iLSTD, because the latter only updates the prioritized components of w, and neglects the experience corresponding to the left $K-m$ components. The RMS error of MR approaches that of LSTD much faster than iLSTD/$m-$iLSTD. The reason may be that the step-size of iLSTD cannot guarantee that the new residual error is always smaller than the old one. MR is also advantageous in that it has no parameter to select by hand.

The computational efficiency of each algorithm was measured by its CPU time per time step. The running time with a variety of problem sizes was tested on a Pentium(R) 4 PC (Intel(R) CPU 3.00GHZ; RAM 1.00GB).

Figure 4 shows that LSTD is very efficient for small problem (*e.g.*, $K \leq 26$), because its matrix has small dimensions and is easy to invert. However, the running time of LSTD

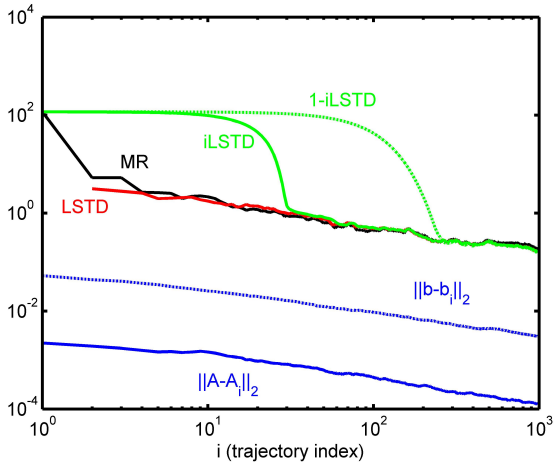

Figure 3: Modeling errors and RMS errors on Boyan chain with 101 states (26 features). Both axes use Log scale.

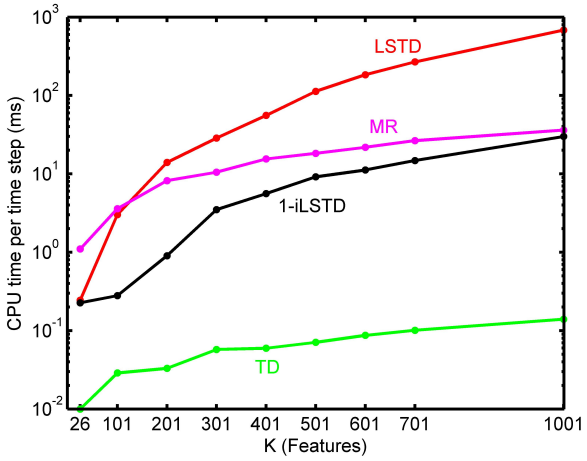

Figure 4: Comparison of CPU time per time step. Each point in the figure is averaged over one trajectory.

grows much more quickly than the other algorithms as the size of problem increases. TD is the most computationally efficient. The running time of MR and $1-$iLSTD occupies in the middle of LSTD and TD. Although MR is a little more complex than $1-$iLSTD, it increases more slowly than $1-$iLSTD with respect to the problem size. When the problem size increases to some scale, the difference becomes fairly narrow.

CSR format enables MR to manipulate much smaller size of data at each time step than LSTD. As shown by Figure 5, on Boyan chain with 801 states, LSTD has to manipulate $201 \times 201 = 40401$ entries of A_i at each time step. In contrast, CSR needs only $800+800+202 = 1802$ memory. Thus MR manipulates only 1802 elements of $\mathbb{S}_A^t$ at each time step. In fact, only the entries of array a_i are updated after a few trajectories, because no new nonzero entries of A_i emerges, and arrays c_i and d_i remain unchanged.

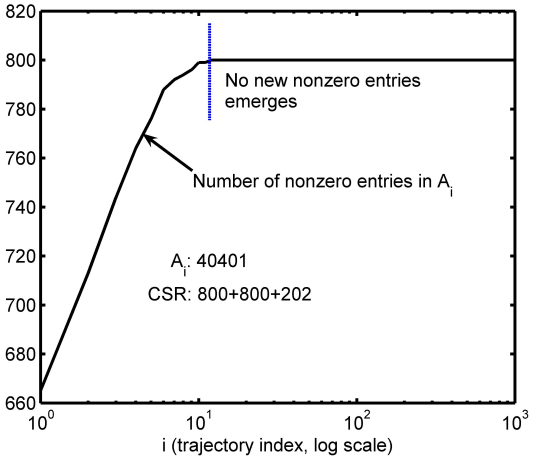

Figure 5: Comparison of the sizes of A_i and CSR on Boyan chain with 801 states (201 features). An entry in A_i is treated as 0 if its absolute value is smaller than 0.000001.

6 Discussion

MR algorithm avoids the need to tune parameters. As a big concern, people wonder if any other assumption is introduced by MR. Actually MR algorithm is developed on no more assumption than previous algorithms. We are very interested whether the step-size can perform well on complex RL problems. Another interesting question is, are RL tasks we encounter usually sparse? As an ongoing investigation, we found that in cart-pole balancing tasks, matrix A_i is very sparse. This may be a positive indication that CSR can be used for these continuous control problems that are solved by discretization.

7 Acknowledgement

We thank Li Lihong and one reviewer for pointing out that the rule of step-size (20) minimizes the new residual error. Li Lihong also helpfully discuss an initial version of this paper with the first author. We thank three anonymous reviewers for helping improve on many aspects of the paper.

References

Bertsekas, D. P., and Tsitsiklis, J. N. (1996) *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA.

Boyan, J. A. (1999) Least-squares temporal difference learning. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pp. 49-56. Morgan Kaufmann, San Francisco, CA.

Boyan, J. A. (2002) Technical update: Least-Squares Temporal Difference Learning. *Machine Learning* **49**(2-3):233-246.

Bradtke, S., and Barto, A. G. (1996) Linear least-squares algorithms for temporal difference learning. *Machine Learning* **22**(1-3):33-57.

Geramifard, A., Bowling, M., and Sutton, R. S. (2006) Incremental Least-Squares Temporal Difference Learning. In *Twenty-First*

National Conference on Artificial Intelligence (AAAI-06), pp. 356-361.

Lagoudakis, M., and Parr, R. (2003) Least-squares policy iteration. *Journal of Machine Learning Research* **4**: 1107-1149.

Moore, A. W., and Atkeson, C. G. (1993) Prioritized Sweeping: Reinforcement Learning with Less Data and Less Real Time. *Machine Learning* **13**:103-130.

Nedić, A., and Bertsekas, D. P. (2003) Least-Squares Policy Evaluation Algorithms with Linear Function Approximation. *Discrete Event Systems: Theory & Applications* **13**:79-110.

Saad, Y. (2003) *Iterative Methods for Sparse Linear Systems,* 2*nd edition*. SIAM, Philadelphia, PA, USA.

Sutton, R. S. (1988) Learning to Predict by the Methods of Temporal Differences. *Machine Learning* **3**:9-44.

Sutton, R. S., and Barto, A. G. (1998) *Reinforcement Learning: An Introduction*. Cambridge MA, MIT Press.

Tadić, V. (2001) On the Convergence of Temporal-Difference Learning with Linear Function Approximation. *Machine Learning* **42**(3):241-267.

Tsitsiklis, J. N., and Van Roy, B. (1997) An Analysis of Temporal-Difference Learning with Function Approximation. *IEEE Transactions on Automatic Control* **42**:674-690.