Investigating the maximum likelihood alternative to TD(λ)

Fletcher Lu F2LU@CS.UWATERLOO.CA
Relu Patrascu RPATRASC@CS.UWATERLOO.CA
Dale Schuurmans DALE@CS.UWATERLOO.CA
Dept. of Computer Science, University of Waterloo, 200 University Ave. W., Waterloo, ON, N2L 3G1 Canada

Abstract

The study of value estimation in Markov reward processes has been dominated by research on temporal difference methods since the introduction of TD(0) in 1988. Temporal difference methods are often contrasted with a maximum likelihood approach where the transition matrix and reward vector are estimated explicitly and converted into a value estimate by solving a matrix equation. It is often asserted that maximum likelihood estimation yields more accurate values, but the temporal difference approach is far more efficient computationally. In this paper we show that the first assertion is true, but the second can be false in many circumstances. In particular, we show that a reasonable implementation of a sparse matrix solver can yield run times for maximum likelihood that are competitive with TD(λ). In our experiments the maximum likelihood estimator yields more accurate values. This higher accuracy in conjunction with competitive execution time suggests that a model based approach might yet be worth pursuing in scaling up reinforcement learning.

1. Introduction

Estimating the expected future reward in a Markov reward process is fundamental to many approaches for reinforcement learning and Markov decision process planning. This assertion is especially true of methods that employ a policy iteration approach and therefore require an estimate of the value for the current policy at each iteration. In this paper we focus on value estimation and consider alternative estimators that have differing advantages for this task. In particular, we will investigate the problem of estimating the expected sum of future rewards in an infinite horizon discounted Markov reward process (Sutton & Barto, 1998).

The problem of value estimation in Markov reward processes has been vigorously studied, and a lot of attention has been paid to temporal difference estimators (Sutton, 1988; Dayan, 1992; Sutton & Barto, 1998). Temporal difference estimators are computationally efficient methods that have an appealing bootstrap nature, where the value estimate for a visited state is updated to reflect both the immediate reward and the estimated future values currently held in subsequent states. Temporal difference estimation is often called the *direct* or non-model based approach because these methods attempt to directly estimate the value function, and do not otherwise build an explicit model of the state transition matrix or the average reward vector. Another direct approach is Monte Carlo (Barto & Duff, 1993) which we compare below.

In contrast with these direct estimators, the maximum likelihood approach follows an *indirect* or model based strategy where an explicit estimate of the state transition matrix and average reward vector is first computed, and then the value estimate is recovered by solving a matrix equation. It is the computational complexity of solving this matrix equation that is often held as the primary reason for precluding maximum likelihood as a practical estimator for these tasks, since it appears to fundamentally require more time and space (Singh & Sutton, 1996). (We discuss other reasons below.) In this paper we show that maximum likelihood need not be inefficient, even in common experimental settings, if one takes care to exploit the sparse structure of the matrices that often arise in these problems. The benefit of maximum likelihood estimation, which we verify below, is that it almost always yields more accurate estimates than either the temporal difference or Monte Carlo methods. Therefore, it seems that explicit model based approaches such as maximum likelihood may be worthwhile to pursue in more significant large scale studies.

In this paper, we focus on comparing estimators in terms of their estimation accuracy given comparable data resources, and their computational efficiency in terms of time and space requirements. We will not focus on other issues such as on-line versus off-line algorithmic structure, or model based versus non-model-based algorithmic structure. However before continuing, we first note that one could assess the computational efficiency of the value estimation methods under two regimes: *single shot*, where one considers the computational resources needed to produce a single estimate of the value function given a batch of data; and *incremental*, where one considers the computational resources needed to produce a complete value function at every step during exploration. Clearly, in the latter case temporal difference estimators retain their computational advantage, and this is perhaps one reason why maximum likelihood techniques are not often used. Nevertheless, some progress can be made toward closing the gap. In this paper we begin to make progress by considering the first, simpler analysis.

2. Preliminaries

A discrete time Markov reward process on a finite set of N states, $n = 1, ..., N$ is described by a transition model $\mathrm{P}(S_{i+1} = m | S_i = n)$, where we assume the transition probabilities do not change over process time i (stationarity assumption). Such a transition model can be represented by an $N \times N$ matrix P, where $P(n, m)$ denotes $\mathrm{P}(S_{i+1} = m | S_i = n)$ for all process times i. The reward R_i observed at time i is independent of all other rewards and states given the state S_i visited at time i. We also assume the reward model is stationary and therefore let $\mathbf{r}(n)$ denote $\mathrm{E}[R_i | S_i = n]$ and $\boldsymbol{\sigma}^2(n)$ denote $\mathrm{Var}(R_i | S_i = n)$ for all process times i. Thus, $\mathbf{r}$ and $\boldsymbol{\sigma}^2$ represent the vectors (of size $N \times 1$) of expected rewards and reward variances respectively over the different states $n = 1, ..., N$.

We consider an infinite horizon process where the initial state S_0 is drawn from an initial distribution, which can be represented by an $N \times 1$ vector $\mathbf{p}_0$ such that $\mathbf{p}_0(n) = \mathrm{P}(S_0 = n)$. Once S_0 is chosen, state transitions $S_{i-1} \to S_i$ are performed where each destination state S_i emits a reward $S_i \to R_i$. Therefore, the overall structure of the probability model is that of a hidden Markov model. We will let n, m, ℓ refer to states respectively ($1 \leq n, m, \ell \leq N$), and i, j, k refer to process times ($0 \leq i, j, k$). Thus, a stationary, discrete time, discounted infinite horizon Markov reward process will be completely specified by a discount factor γ, an $N \times N$ matrix P, an $N \times 1$ vector $\mathbf{p}_0$, and the reward distribution, which can be summarized by the $N \times 1$ vectors $\mathbf{r}$ and $\boldsymbol{\sigma}^2$.[1]

[1]Note that an analysis of expected squared prediction error, MSE, requires no further information about the distribution of rewards other than their means and variances.

The value function $\mathbf{v}(n)$ is defined to be the expected sum of discounted future rewards obtained by starting in a state $S_0 = n$. That is, $\mathbf{v}$ is a vector given by

$$\begin{aligned} \mathbf{v} &= \mathbf{r} + \gamma P \mathbf{r} + \gamma^2 P^2 \mathbf{r} + \cdots \\ &= \mathbf{r} + \gamma P \mathbf{v} \end{aligned} \quad (1)$$

Therefore, if P and $\mathbf{r}$ are known then $\mathbf{v}$ can be calculated explicitly by solving the matrix equation

$$(I - \gamma P)\mathbf{v} = \mathbf{r} \quad (2)$$

All of the estimators we consider will produce estimates $\hat{\mathbf{v}}$ of the value function by processing sample trajectories that have been generated by some independent sampling strategy. The specific sampling strategy we consider depends on whether or not the Markov reward process has an absorbing state.

Absorbing restarts If the process has an absorbing state (and finite expected walk length) then the sampling process produces independent trajectories by restarting at a state drawn from the initial distribution $\mathbf{p}_0$ whenever the absorbing state is reached.

Random walk If the Markov reward process does not have an absorbing state (and is irreducible) then we sample one long trajectory through the reward process.

Several estimators can be applied to the value estimation problem in Markov reward processes. These methods attempt to estimate the value of each state by processing sampled trajectories. The specific estimators we consider are: maximum likelihood, Monte Carlo, and TD(λ).

Maximum likelihood For the individual parameters $P(n, m)$ and $\mathbf{r}(n)$, the maximum likelihood estimates are given by

$$\begin{aligned} \hat{P}(n, m) &= \frac{\#\{i : s_i = n \text{ and } s_{i+1} = m\}}{\#\{i : s_i = n\}} \\ \hat{\mathbf{r}}(n) &= \frac{\sum_{\{i : s_i = n\}} r_i}{\#\{i : s_i = n\}} \end{aligned}$$

if $\#\{i : s_i = n\} > 0$ (otherwise undefined). Here $\#$ denotes set cardinality. Given these quantities one can obtain the maximum likelihood estimate simply by plugging $\hat{P}$ and $\hat{\mathbf{r}}$ into Equation 2 and solving for the vector $\hat{\mathbf{v}}_{ml}$ in

$$(I - \gamma \hat{P})\, \hat{\mathbf{v}}_{ml} = \hat{\mathbf{r}} \quad (3)$$

Solving this equation is perceived to be the most arduous aspect of producing an ML estimate, since it can require $O(N^3)$ run time using standard algorithms. Nevertheless, ML yields a consistent estimator in the sense that $\lim_{T\to\infty} \hat{\mathbf{v}}_{ml} \to \mathbf{v}$ with probability one for reachable states, since both $\hat{P} \to P$ and $\hat{\mathbf{r}} \to \mathbf{r}$ by the strong law of large numbers (Ash, 1972). However, ML is actually *biased*; that is, generally, $\mathrm{E}[\hat{\mathbf{v}}_{ml}] \neq \mathbf{v}$.[2] However, despite this bias ML yields a good estimator for $\mathbf{v}$ because it tends to make efficient use of the sample data by estimating the transition probabilities $\mathrm{P}(S_{i+1} = m|S_i = n)$ in terms of every visit to $S_i = n$ regardless of process time i. We empirically verify below that it does indeed yield superior estimates. In addition, ML requires $\mathrm{O}(t)$ space where t is the number of nonzeros in the matrix $I - \gamma\hat{P}$ of Equation 3.

Monte Carlo The on-line Monte Carlo estimator (MC) first initializes a default guess $\hat{\mathbf{v}}_{mc} = \mathbf{g}$, and then upon visiting a state n at process time i and receiving the future discounted reward sequence $r_{i,n} = R_i + \gamma R_{i+1} + \cdots$, it updates the value estimate for n according to $\hat{\mathbf{v}}_{mc}(n) \leftarrow (1-\alpha)\hat{\mathbf{v}}_{mc}(n) + \alpha r_{i,n}$, (Singh & Sutton, 1996; Singh & Dayan, 1998). (We will use a constant α for simplicity.) MC estimators are obviously computationally efficient. If T is the total number of state-to-state transitions observed, then MC produces a value estimate in $O(T)$ time and $O(N)$ space. However, note that MC does not exploit the fact that the Markov process is stationary: The MC estimator could just as easily be applied to a nonstationary process without modification. This suggests that MC is an inefficient estimator for stationary reward processes, which we verify below.

TD(λ) Finally, we consider the temporal difference estimator TD(λ), which is conventionally implemented using eligibility traces, as shown in Figure 1 (Sutton & Barto, 1998). Subtle variants of this procedure are obtained by changing the way the eligibilities are updated. For example, if n is the current state and $\ell \neq n$, then all procedures use the update $\mathbf{e}(\ell) \leftarrow \gamma\lambda\mathbf{e}(\ell)$, however for state n "accumulate trace" uses the update $\mathbf{e}(n) \leftarrow \gamma\lambda\mathbf{e}(n) + 1$ whereas "replace trace" uses the update $\mathbf{e}(n) \leftarrow 1$ (Singh & Dayan, 1998). TD(λ) is also perceived to be computationally efficient, as it runs in $O(TN)$ time, in the worst case, while requiring $O(N)$ space. Although these crude analyses suggest that ML might always be computationally more expensive, we will see below that this need not always be the case.

[2] This can be seen by noting that even though $\mathrm{E}[\hat{P}^j\hat{\mathbf{r}}] = \mathrm{E}[\hat{P}^j]\mathrm{E}[\hat{\mathbf{r}}]$, $\mathrm{E}[\hat{\mathbf{r}}] = \mathbf{r}$ and $\mathrm{E}[\hat{P}] = P$ (since R_i is independent of S_k given $S_i = n$), it is not true that $\mathrm{E}[\hat{P}^j]$ is equal to P^j in general. Consider the special case of determining $\mathrm{E}[\hat{P}^2(n,m)]$. Here $\hat{P}^2(n,m) = \sum_{\ell=0}^{N-1} \hat{P}(n,\ell)\hat{P}(\ell,m)$, where for terms such that $\ell \neq n$ we have $\hat{P}(n,\ell)$ independent of $\hat{P}(\ell,m)$, as desired. However for the term $\ell = n$, the quantities $\hat{P}(n,n)$ and $\hat{P}(n,m)$ are *not* independent. For example, in the case where $m = n$ they become $\hat{P}(n,n)^2$, whose expectation is given by $\mathrm{E}[\hat{P}(n,n)^2] = \mathrm{Var}(\hat{P}(n,n)) + (\mathrm{E}[\hat{P}(n,n)])^2 \;>\; (\mathrm{E}[\hat{P}(n,n)])^2$.

```
Initialize v̂_td(n) = g(n), e(n) = 0, for all 1 ≤ n ≤ N
Repeat for each trajectory:
   Draw an initial state n according to p_0
   Repeat for each step of trajectory:
      Observe next state m and reward r
      δ ← r + γ v̂_td(m) − v̂_td(n)
      For all states ℓ:
         v̂_td(ℓ) ← v̂_td(ℓ) + α δ e(ℓ)
         e(ℓ) ← update-eligibility(ℓ)
      n ← m
   Until state n is terminal
```

Figure 1. On-line TD(λ) with eligibility traces

3. Assessing estimation accuracy

First we consider some brief experimental results to confirm the well known folklore surrounding the statistical accuracy and computational efficiency of maximum likelihood and temporal difference estimators (Singh & Sutton, 1996; Singh & Dayan, 1998). We conducted a series of experiments on artificial domains where we could control the number of states N, the number of state-to-state transitions sampled T, and the number of unique state-to-state transition pairs observed t' (which determines the sparsity, $t = t' + N$, of the matrix $\hat{P}$ used by ML). The four domains we considered are described in Figure 2.

To assess the accuracy of the various estimators on these problems we measure their expected squared prediction error (MSE) relative to the true value function $\mathbf{v}$. Since the estimate $\hat{\mathbf{v}}$ and target values $\mathbf{v}$ are both vectors, we report a summary MSE score which is the maximum of the individual MSEs over each component (state). Run time is measured in terms of the total time to process the sample data and produce the final estimate.

Results shown in Tables 1 and 2 support the superior quality of ML as a statistical estimator. In our experiments ML uniformly obtains the smallest MSE values, regardless of problem size N, sample size T or transition matrix density t; see Table 1. Interestingly, ML also produces its value estimates in reasonable time compared to the other approaches; as shown in Table 2. These run times were recorded for simple Matlab implementations of the different procedures. What is apparent from Table 2 is that ML's cubic run time in

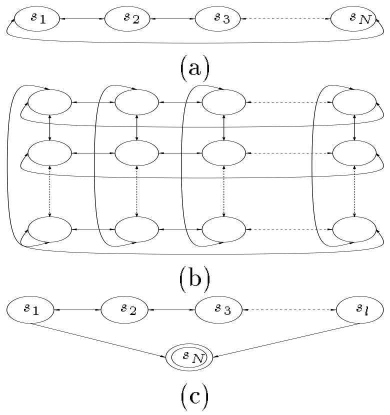

Figure 2. (a) Ring problem and (b) Torus problem: random probability of moving to neighboring states, random rewards between 0 and 1; (c) sd problem as defined in Figure 2 of (Singh & Dayan, 1998); the random problem, not shown here: entries in P generated uniformly at random and then renormalized; random rewards between 0 and 1.

The Random Problem

T	**ML**	MC1	MC2	TDR	TDA	t'
10	**62.86**	75.96	76.17	74.76	75.28	9
100	**6.493**	8.680	11.48	64.11	59.97	92
1000	**0.046**	1.172	0.143	14.23	7.974	538

The Ring Problem

T	**ML**	MC1	MC2	TDR	TDA	t'
10	**180.5**	200.3	200.3	199.9	198.3	5
100	**148.8**	168.7	168.7	180.8	179.6	13
1000	**113.6**	127.4	127.4	151.7	148.1	32

The Torus Problem

T	**ML**	MC1	MC2	TDR	TDA	t'
10	**149.5**	169.2	169.2	159.5	161.1	8
100	**135.8**	145.9	145.9	154.8	154.3	47
1000	**63.5**	72.4	72.5	132.7	131.8	98

The SD Problem

T	**ML**	MC1	MC2	TDR	TDA	t'
10	**0.524**	1.000	1.000	2.274	2.292	6
100	**0.436**	0.820	0.820	2.477	2.343	28
1000	**0.036**	0.053	0.045	1.796	1.697	51

Table 1. Mean Square Error. $N = 30$. **Legend ML:** Maximum Likelihood; **MC1:** 1st-visit MC; **MC2:** every-visit MC; **TDR:** TD(λ) with replacing eligibility traces; **TDA:** TD(λ) with accumulating eligibility traces. The results for best empirically found setting of λ in each case.

The Random Problem

N	ML	MC1	MC2	TDR	TDA	t'
5	0.118	0.003	0.231	0.322	0.357	25
10	0.121	0.007	0.232	0.489	0.527	92
20	0.129	0.013	0.235	0.822	0.863	326
30	0.137	0.020	0.238	1.155	1.199	534
50	0.148	0.033	0.243	1.817	1.866	778
100	0.169	0.064	0.255	3.472	3.545	937
250	0.254	0.148	0.289	8.623	8.397	991

The Ring Problem

N	ML	MC1	MC2	TDR	TDA	t'
5	0.117	0.003	0.229	0.322	0.354	10
10	0.118	0.006	0.231	0.487	0.519	19
20	0.121	0.012	0.234	0.820	0.850	36
30	0.124	0.017	0.236	1.152	1.181	46
50	0.130	0.021	0.240	1.818	1.843	50
100	0.145	0.025	0.249	3.482	3.498	24
250	0.221	0.059	0.278	8.418	8.410	50

Table 2. Run time in seconds. $T = 1000$. Same legend as above.

N is not always clearly revealed in real experiments. Below we show that in many natural circumstances, the run time of ML is in fact far better than cubic in N, explaining why not only its accuracy, but also its run time can be superior to TD(λ) in many experimental settings.

4. The computational efficiency of ML

An efficient implementation of the ML estimator must address the problem of solving the linear matrix equation, Equation 3, for an estimated transition model $\hat{P}$ and reward vector $\hat{\mathbf{r}}$. Let

$$A \quad = \quad I - \gamma \hat{P}.$$

Note that A is a square nonsingular matrix, and therefore Equation 3 may be solved by a number of factorization methods. Since factorization typically is considered to run in $\mathrm{O}(N^3)$ time (Golub & Van Loan, 1989), most applications forego this approach for the faster $\mathrm{O}(TN)$ approach of TD(λ) and Monte Carlo methods.[3] However, the factorization run time is a gross generalization assuming a dense system using a direct solver and assuming a linear relation between T and N. In the remainder of this section, let $\mathbf{v} = \hat{\mathbf{v}}_{ml}$ and note that Equation 3 becomes

$$A\mathbf{v} = \mathbf{r} \tag{4}$$

Here A is an $N \times N$ matrix. Since t is the number of nonzeros in the matrix A, then

$$N \leq t \leq N^2. \tag{5}$$

Clearly T and N are not simply linearly related. This paper will demonstrate that we can often achieve an $\mathrm{O}(TN)$ run time to solve Equation 4 using a combination of careful analysis and various matrix solve approaches.

We now consider the cases under which it is possible to solve $A\mathbf{v} = \mathbf{r}$ in $\mathrm{O}(TN)$ run time. First note that, under the special case of $t = O(T^{1/2})$, a direct factorization of matrix A runs in at worst $O(t^3)$ time when $t = N$, and therefore $O(t^3) \leq O(T^{3/2}) = O(TN)$. So when $t = O(T^{1/2})$ a matrix solve has comparable efficiency to TD(λ). In fact, when $t < T^{1/2}$ a matrix solve is faster than TD(λ). Intuitively, these situations can be interpreted as, when the TD(λ) method starts repeating state-to-state transitions sufficiently often during their sample runs, a matrix solve approach then becomes as fast or faster in determining a solution.

[3] A large body of factorization algorithms exist. However, we simply refer to factorization algorithms since they all have a worst case runtime of $\mathrm{O}(N^3)$, when not considering any other factors such as matrix density or structure.

Under the remaining conditions of $T^{1/2} < t \le T$, we can divide the problem into three cases: (1) *dense*: $c_1 N^2 \le t \le N^2$, (2) *intermediate*: $N + c_2 N^{2/3} < t < c_1 N^2$, and (3) *sparse*: $N \le t \le N + c_2 N^{2/3}$; where c_1 and c_2 are constants. Intuitively, a dense matrix has nonzeros in most of its element positions. Conversely, a sparse system has few nonzeros in its element positions and an intermediate matrix lies between a dense and sparse matrix. Of these cases, the dense and sparse situations are the easiest to show competitive computational efficiency for matrix solve. However, the intermediate case poses remaining challenges. We outline some ideas for tackling this difficult situation below.

4.1 Dense case

Lemma 4.1 *Given $c_1 N^2 \le t \le N^2$, a naive matrix solve runs in $O(TN)$ time.*

Proof The factorization time of a matrix solve runs in $O(N^3)$ for an $N \times N$ matrix. By assumption we have $c_1 N^2 \le t \le N^2$ and therefore $t^{1/2} \le N \le c_1^2 t^{1/2} \le c_1^2 T^{1/2}$. Thus $N^2 = O(T)$ and hence $N^3 = O(TN)$. ∎

So under a very dense system, the matrix equation for ML estimation can be solved as quickly as running through the TD(λ) estimator.

4.2 Sparse case

Here we assume $N \le t \le N + c_2 N^{2/3}$. Note that since γ in Equation 3 is a discount factor, we have $\gamma < 1$. Also, since $\hat{P}$ is a probability matrix, $\sum_{m=1}^{N} \hat{P}(n,m) = 1$ for all n and therefore $\gamma \sum_{m=1}^{N} \hat{P}(n,m) = \gamma$ for all n. Thus the matrix $A = I - \gamma \hat{P}$ is diagonally dominant. We can therefore symmetrically pivot our matrix during factorization with respect to nonzero fill only, while maintaining stability. To do so let us symmetrically permute the rows and columns of the matrix A in the following manner. First let $n = N$. Then repeat the following steps until $n = 1$:

- Search rows $1..n$ for row m with the most nonzeros.
- Swap row m with row n, column m with column n.
- Set $n \leftarrow n - 1$.

We will call this new permuted matrix $\hat{A}$.

Lemma 4.2 *The number of nonzeros in any of the first $N - c_2 N^{2/3}$ rows of $\hat{A}$ has at most one nonzero in the off diagonal.*

Proof is by contradiction. Suppose a row in the first $N - c_2 N^{2/3}$ rows has more than one nonzero. Then the last $c_2 N^{2/3}$ rows all have at least two nonzeros in each row, by the construction of $\hat{A}$. Therefore, the last $c_2 N^{2/3}$ rows contain at least $2c_2 N^{2/3}$ nonzeros and at least one row in the first $N - c_2 N^{2/3}$ rows has at least 2 nonzeros. This situation produces a current total of $2c_2 N^{2/3} + 2$ nonzeros. There is exactly one nonzero in each row not including the ones on the diagonal contributed by the identity matrix in the matrix $A = I - \gamma \hat{P}$. (This situation is due to how $\hat{P}$ was constructed, since it is a probability matrix where a row/column is only formed if a nonzero was produced by a state-to-state transition.) Therefore $\hat{A}$ has at least one nonzero in each of the remaining $N - c_2 N^{2/3} - 1$ rows for a total of:

$$
\begin{aligned}
&= 2c_2 N^{2/3} + 2 + N - c_2 N^{2/3} - 1 \\
&= N + c_2 N^{2/3} + 1 \\
&> N + c_2 N^{2/3}
\end{aligned}
$$

nonzeros. Since t is the number of nonzeros in A, this result contradicts the fact that $t \le N + c_2 N^{2/3}$. ∎

Lemma 4.3 *Given $N \le t \le N + c_2 N^{2/3}$, matrix Equation 4 can be solved in $O(TN)$ time.*

Proof To solve a matrix equation one can generally consider an LU factorization approach, which is attractive because of its lack of any assumption on structure (Duff et al., 1986). Here A is factored into a lower triangular matrix L and an upper triangular matrix U such that $A = LU$. This transforms the original problem of solving $A\mathbf{v} = \mathbf{r}$ into first solving $L\mathbf{y} = \mathbf{r}$ and then solving $U\mathbf{v} = \mathbf{y}$. Since L and U are both triangular, these equations can be solved by back substitution in quadratic time. However, the cost of calculating the LU factorization is $O(N^3)$ in the worst case. This is because the factorization $A = LU$ is calculated by an iterative process, where for an $k \times k$ matrix A a single factorization step is:

$$
\begin{aligned}
A_{k\times k} &= \begin{bmatrix} a & \mathbf{w}^\top \\ \mathbf{u} & A_{k-1\times k-1} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ \frac{\mathbf{u}}{a} & I \end{bmatrix} \begin{bmatrix} a & \mathbf{w}^\top \\ 0 & A_{k-1\times k-1} - \frac{\mathbf{u}\,\mathbf{w}^\top}{a} \end{bmatrix}
\end{aligned}
$$

Note that $\mathbf{u}\,\mathbf{w}^\top$ is the outer product. If vectors $\mathbf{u}$ and $\mathbf{w}$ are both full (i.e. no nonzero entries) then the outer product is an $(k-1)^2$ product. Thus, the complete factorization involves factoring the subsequent $k-1 \times k-1$ submatrix $A_{k-1\times k-1} - \frac{\mathbf{u}\,\mathbf{w}^\top}{a}$ in the same manner as the $A_{k\times k}$ matrix, *etc.* Each factorization step builds the lower triangular matrix L from the column vectors $\frac{\mathbf{u}}{a}$ and the upper triangular matrix U is built from the row vectors $\mathbf{w}^T$. The complete factorization produces a quadratic summation series whose total is cubic.

However, in the case we are considering in this subsection one can exploit sparsity to reduce the computational complexity of factoring. First, note that

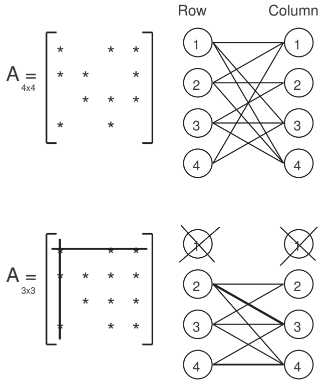

Figure 3. Elimination in a bipartite graph of a matrix A.

nonzero fill during factorization can be represented by a bipartite graph, where each factorization step eliminates one pair of nodes $\{n, m\}$. After such a step, every node that was connected to node n then becomes connected to every node that was connected to node m. Thus the degree of node n multiplied by the degree of node m represents the total number of multiplications performed at this iterative step; see Figure 3.

Now recall that for matrix $\hat{A}$, the first $N - c_2N^{2/3}$ row nodes have at most one arc connected to a column node (not including the diagonal arc connected to the opposite column node). So at each step all the row nodes that have only one arc will continue to have only one arc, since they only form a new connection if the one column node they were originally connected to is eliminated also. Therefore, during the first $N - c_2N^{2/3}$ factorization steps, at most

$$\begin{aligned}(N-1) &+ (N-2) + \ldots + N - c_2N^{2/3} \\ &= \frac{N^2 - N}{2} - \frac{(N - c_2N^{2/3})(N - c_2N^{2/3} - 1)}{2} \\ &= c_2N^{5/3} - c_2^2\frac{N^{4/3}}{2} - c_2\frac{N^{2/3}}{2}\end{aligned}$$

multiplications are performed. During the last $c_2N^{2/3}$ factorization steps a matrix of size $c_2N^{2/3} \times c_2N^{2/3}$ is being factored. Therefore the run time is at worst $\mathrm{O}((N^{2/3})^3) = \mathrm{O}(N^2)$. Thus, the total run time for the factorization and solution is $O(N^2) \leq O(TN)$. ∎

Note that the point of this result is not simply that one can naively run a generic linear equation solver and achieve competitive run times in this case. The point is that by employing techniques for sparse linear systems and analyzing the effects of the presumed sparsity on their efficiency, one can recover provably competitive run times for the maximum likelihood approach.

4.3 Intermediate case

The case of intermediate matrix density is the situation where the assumption that a direct solve approach is less efficient than TD(λ) truly holds. Under a direct solve approach, a naive method that does not take into account structure would produce a cubic run time in general. However, even under this situation we can under certain circumstances obtain a quadratic run time in N by: (1) using iterative solve methods (Hageman & Young, 1981; Young, 1971); (2) extending our sample runs to see if a localization of substates occurs that moves us into one of the other cases; or (3) inverting the matrix by a Monte Carlo method.

First, the efficiency of using iterative methods to solve linear systems depends on the eigenvalues of the matrix. Since the convergence rates of iterative methods depend on the numerical values of the matrix, they cannot always guarantee a quadratic solve time. However one can often shift the eigenvalues to increase the rate of convergence. This shift unfortunately usually reduces the accuracy of the solution. The body of research in this area is extensive. However, the dependence on the numerical values for convergence rates limits the effectiveness of this approach in comparison to the TD(λ) alternative.

Second, unless a system is completely evenly distributed, some subset of states will generally be visited more often than others. Such a localization of activity should be observed during sufficiently large sample runs. Therefore, under some systems, if we simply continue our sampling long enough we may "push" our system into one of the alternate sparse or dense cases.

The third approach exploits a method of matrix inversion technique introduced by Von Neumann and Ulam (Forsythe & Leibler, 1950), which uses a random walk to determine the inverse of a matrix. Once the inverse is found, the system can simply be solved by multiplying the inverse with the vector $\mathbf{r}$ of Equation 4. This method uses a sequence of multiplications formed through a random walk to determine an individual element of the matrix being inverted. Since the complete matrix has N^2 elements, the approach needs to run in at least N^2 walks to get a good approximation of the inverse matrix. However, these random walks can be run in parallel, so the number of random walks does not have to directly relate to the running time of this approach. Similar to the iterative method, the running time for determining a single element is dependent on the random walk time which in turn depends on the numerical values of the system and therefore renders the same drawback as the iterative method. A second drawback to this approach is that the inverse solution is an expected value. In order to reduce the inherent variance in the approach the random walks should be repeated.

5. Network Structure

The previous section demonstrated cases where the ML direct solve approach can perform efficiently in comparison with TD(λ), regardless of network structure. Here we demonstrate that for acyclic networks the direct solve approach will always be faster. Also, for cyclic networks, we provide a parameter bound whereby if the number of state-to-state transitions T exceeds that parameter, then ML is faster.

5.1 Acyclic Networks

Lemma 5.1 *Given a Markov reward process with no cycles in the state space, the ML estimate can be calculated in few flops than the TD(λ) estimate, in the big-O sense.*

Proof As stated by Singh and Sutton (1996), each step of temporal differencing with eligibility traces runs in $O(N)$ flops, where N is the number of states in the system. The worst case run time is $O(TN)$ where T is the number of steps taken during sampling runs. Note that $T \geq N$ since only states visited can be assigned a value in the temporal difference method. As illustrated in Figure 1 any temporal difference with eligibility trace will require at least one addition and several multiplications to update each state value for each step in the sampling runs. Therefore the run time of TD(λ) is actually cTN for $c > 1$.[4] Since an acyclic network forms a DAG we can choose a labeling for the states whereby we number each state n, $1 \leq n \leq N$, such that for any state-to-state transition from n to m we have $n < m$. If we consider the resulting probability transition matrix $\hat{P}$ of such a network, then $\hat{P}$ is a strictly upper triangular matrix. In terms of the resulting equation that needs to be solved to value the states, the matrix $I - \gamma\hat{P}$ of Equation 3 is also upper triangular. Therefore, the only operations needed to solve Equation 4 are back substitutions. The back substitution equation for finding the value of a single state n, in terms of Equation 4, would be:

$$\mathbf{v}(n) = \frac{\mathbf{r}(n) - \sum_{m=N}^{n+1} A(n,m)\mathbf{v}(m)}{A(n,n)} \tag{6}$$

(Note that each $\mathbf{v}(n)$ in the summation should already be known from previous back substitutions.) Equation 6 shows that the number of additions performed is equal to the number of nonzeros in row n of the probability matrix $\hat{P}$. If we let x_n be the number of nonzeros in row n, then by similar analysis, Equation 6 requires $x_n + 1$ multiplications. Therefore, the total number of flops required to find a solution to Equation 4 is $O(N^2)$. However the constant coefficient of this quadratic run time is actually one.[5] Therefore, TD(λ) runs in $O(TN)$ time but the value of the constant c hidden in the big-O notation is greater than one. ML runs in $O(N^2)$ flops. ML's constant coefficient for the N^2 term is exactly one. Thus for acyclic networks, ML is faster than TD(λ). ∎

5.2 Cyclic Networks

Although it is true that standard matrix solvers run in cubic time in the worst case, the actual constant coefficient of the cubic term is 1/3 (Golub & Van Loan, 1989). From the analysis of the previous section the run time of TD(λ) based on the number of arithmetic operations is cTN, where $c > 1$. Therefore, if we base our preference for using temporal differencing on its superior execution time, then the number of steps performed during sampling must be $T < \frac{N^2}{3c}$. Since $T \geq N$ (otherwise the state would not be included in our solution), we must visit each state less than $\frac{N}{3c}$ times on average. If we are interested in how accurate TD(λ)'s estimates are, then if the true state-to-state transition matrix is filled with more than $\frac{1}{3c}$ of nonzeros, one of two situations may arise: (1) temporal differencing will not include some possible state-to-state transitions in its sample runs; or (2) if all state-to-state transitions are sampled at least once, then TD(λ) will not be faster than ML in producing its value estimate. Based on the TD(λ) algorithm of Figure 1, a reasonable value of c would be 3. Therefore, if the matrix is more than 11% dense, ML is competitive with TD(λ).

6. Conclusion

This paper has demonstrated that the general assumption that the TD(λ) approach for value estimation in Markov reward processes necessarily achieves better execution time than maximum likelihood is not always true. By using reasonable sparse matrix algorithms with a direct solver, maximum likelihood value estimation can be done as efficiently or more efficiently than the TD(λ) approach in the off-line setting—in cases where exploration yields either a dense or a sparse matrix. In particular, if data collection takes a particularly long time, maximum likelihood will perform

[4] Various temporal differencing algorithms may be able to reduce the number of multiplications, however there will need to be at least one multiplication in order to discount the eligibility by the discount factor γ.

[5] The maximum number of nonzeros in an upper triangular matrix $\hat{P}$ is $N(N-1)/2$.

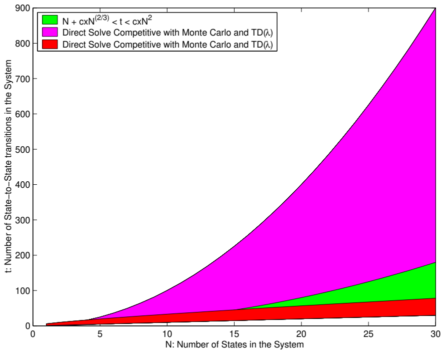

Figure 4. Region where run time of arithmetic operations of Maximum Likelihood competitive with TD(λ).

value estimation faster than TD(λ). Figure 4 illustrates the regions where a direct matrix solve approach is competitive with the TD(λ) approach. The upper region where maximum likelihood is competitive with TD(λ) fills up 89% of the region. Thus the region where TD(λ) appears more efficient is actually quite narrow. For densities less than 11%, many matrix algorithms have been designed to specifically address this region. Moreover, under the case where the reward process forms a DAG over the state space, the ML approach is always faster than TD(λ).

We also experimentally verified the general perception that maximum likelihood provides more accurate value estimates than TD(λ) or MC. Thus, the main advantage of maximum likelihood is that it provides more accurate estimates. However, there are circumstances in which recovering an explicit model of the state transition model by itself might be useful. For example, in situations where designers are attempting to discover the underlying nature of the domain.

An important issue that we did not address in this paper is the efficiency of *approximate* algorithms for both TD(λ) and ML. That is, TD(λ) can be made more efficient at each update by only updating states whose eligibility stays above a threshold tolerance. Such approximation algorithms (Sutton & Barto, 1998; Cichosz, 1995) have been shown to speed up the computation, however at the cost of further solution accuracy. Similarly, for ML estimation, a threshold tolerance could be imposed on the estimated transition matrix $\hat{P}$ and smaller values dropped. This would naturally make $\hat{P}$ sparser and further improve the efficiency of the ML approach. We have not yet analyzed how these two approximations schemes compare to one another in terms of estimation accuracy and computational efficiency. The choice of whether to use an exact or approximate algorithm is determined typically by practical considerations of resources and the desired accuracy of the solution. For this paper, we focus our attention on exact algorithms leaving an analysis of approximate algorithms to future research.

For other future work, we plan to investigate the intermediate case and look into how often such median situations occur in applications. A general set of networks could be identified where the matrix solve method is the desirable approach.

Acknowledgments

Research supported by NSERC and CITO.

References

Ash, R. (1972). *Real analysis and probability.* San Diego: Academic Press.

Barto, A., & Duff, M. (1993). Monte carlo matrix inversion and reinforcement learning. *Proc. NIPS.*

Cichosz, P. (1995). Truncating temporal differences: On the efficient implementation of TD(λ) for reinforcement learning. *J. of A.I. Research*, *2*, 287–318.

Dayan, P. (1992). The convergence of TD(λ) for general λ. *Machine Learning*, *8*, 341–362.

Duff, I. S., Erisman, A. M., & Reid, J. K. (1986). *Direct methods for sparse matrices.* Oxford, Great Britain: Oxford University Press.

Forsythe, G. E., & Leibler, R. A. (1950). Matrix inversion by a Monte Carlo method. *MTAC*, *6*, 127–129.

Golub, G., & Van Loan, C. (1989). *Matrix computations.* Johns Hopkins U. Press.

Hageman, L. A., & Young, D. M. (1981). *Applied iterative methods.* New York: Academic Press.

Singh, S., & Dayan, P. (1998). Analytical mean squared error curves for temporal difference learning. *Machine Learning.*

Singh, S., & Sutton, R. (1996). Reinforcement learning with replacing eligibility traces. *Machine Learning.*

Sutton, R. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction.* Cambridge, Massachusetts: MIT Press.

Young, D. M. (1971). *Iterative solutions of large linear systems.* New York: Academic Press.