Model-Based Least-Squares Policy Evaluation

Fletcher Lu[1] and Dale Schuurmans[1]

School of Computer Science
University of Waterloo
Waterloo, ON, Canada, N2L 3G1
{f2lu, dale}@cs.uwaterloo.ca

Abstract. A popular form of policy evaluation for large MDPs is the least-squares temporal differencing (TD) method. Least-squares TD methods handle large MDPs by requiring prior knowledge *feature vectors* which form a set of basis vectors that compress the system down to tractable levels. Model-based methods have largely been ignored in favour of model-free TD algorithms due to two perceived drawbacks: slower computation time and larger storage requirements. This paper challenges the perceived advantage of the temporal difference method over a model-based method in three distinct ways. First, it provides a new model-based approximate policy estimation method which produces solutions in a faster computation time than Boyan's least-squares TD method. Second, it introduces a new algorithm to derive basis vectors without any prior knowledge of the system. Third, we introduce an iteratively improving model-based value estimator that can run faster than standard TD methods. All algorithms require model storage but remain computationally competitive in terms of accuracy with model-free temporal differencing methods.

1 Background

The study of value estimation in a Markov reward process has been dominated by research in model-free approaches such as Monte Carlo and temporal differencing [1]. The general reasoning behind this preference is that model-free methods are considered to run faster than model-based approaches. Also, avoiding a model saves on storage space. In contrast, model-based methods are generally understood to produce considerably more accurate results. An excellent summary of these advantages and disadvantages was given in [2]. The model-based approach produces the best solution because it forms a maximum likelihood estimate of the value function minimizing approximation error. Recent work has shown that the perception of greater computational efficiency by model-free methods is based on some oversimplifications. A finer analysis has shown that model-based approaches such as maximum likelihood under many circumstances can be computationally competitive with their model-free counterparts [3]. That work principally dealt with polynomial size state spaces.

However, most practical problems deal with exponentially large state spaces. Temporal differencing methods deal with such large MDPs by using a form of compression which reduces the system to a polynomial size based on some set of state assumptions [4]. This form of compression assumes a relation between states whereby some set of states will act in a manner similar to other states. Based on this assumption, least-squares temporal differencing reduces the state space to a tractable problem size. In practical applications, the transition probabilities are formed from a simulation model or from actual experience collected through an agent. The problem with TD methods is that they do not take full advantage of knowledge gained through this data collection. By storing the data in a model, model-based methods can exploit relations not gathered by model-free methods.

This paper exploits a model based approach in several ways. First, based on a model, we provide an alternative least-squares policy evaluation formula to the standard least-squares temporal difference formula (LSTD) introduced in [5] and modified in [4]. We demonstrate that the proposed method runs in faster time than the LSTD method and uses approximately the same storage space. When no model is built from the system, methods such as TD can only form a compression by having prior knowledge which they then exploit. The compression uses a set of basis vectors to reduce the system size. Without prior knowledge the LSTD method cannot be applied. By constructing a model, this paper provides a systematic way to create basis vectors. Assuming that the system can be compressed considerably, we demonstrate that a model-based method of policy evaluation will run faster than temporal differencing. Because it is a model-based method, the value estimates will be more accurate. We further enhance the algorithm to perform iterative improvements in our value estimates allowing early termination for shorter execution time when a certain residual tolerance level is reached.

Although we are able to demonstrate advantages to using a model-based approach to policy evaluation in terms of computational speed and accuracy, greater storage requirements remain a drawback to this method.

2 Least-squares policy evaluation

Table 1. Notation used in complexity analysis

N	Number of states
M	Total number of steps taken over all sampling trajectories
ϕ_i	A basis/feature vector
Φ	Matrix of basis/feature vectors
k	Number of feature vectors. Φ is of size $N \times k$
l	Size of a subset of feature vectors. $l \leq k$.
P	Probability transition matrix
T	Number of nonzeros in P

A discrete time Markov reward process [6] on a finite set of N states, $n = 1, ..., N$ can be 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 assume a stationary reward model. 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 $n = 1, ..., N$.

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

$$\begin{aligned} \mathbf{v} &= \mathbf{r} + \gamma P \mathbf{r} + \gamma^2 P^2 \mathbf{r} + \cdots \\ &= \mathbf{r} + \gamma P \mathbf{v}. \end{aligned} \tag{1}$$

Therefore, if P and $\mathbf{r}$ are known then $\mathbf{v}$ can be solved by

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

The LSTD method modifies equation 2 by using a set of k *feature vectors* ϕ_i each of size $N \times 1$ where $k \leq N$. These feature vectors capture similar properties among states. The LSTD method lets $\Phi \mathbf{w} = \mathbf{v}$ where $\Phi = [\phi_1, ... \phi_k]$. Thus, equation 2 becomes

$$(I - \gamma P)\Phi \mathbf{w} = \mathbf{r}. \tag{3}$$

The matrix $\gamma P \Phi$ is an over-determined matrix. LSTD finds a solution by using a least-squares approach which multiplies both sides of equation 3 by the transpose of Φ

$$\Phi^T (I - \gamma P)\Phi \mathbf{w} = \Phi^T \mathbf{r}. \tag{4}$$

The elegance of this approach, first introduced by Bradtke and Barto [5], is that when performing policy value estimation, it allows for online updating of state values without waiting for termination of data collection in the same way that regular TD does. The complete probability transition matrix is never explicitly represented. Boyan improved the accuracy of the solution by storing a $k \times k$ matrix and iteratively updating that matrix during sampling [4]. Then he solved for the matrix equation of equation 4. Lagoudakis and Parr extended this idea to a policy iteration method [7]. Boyan's LSTD method actually implicitly captures the complete, estimated transition probabilities $\hat{P}$, but just in a compressed $\Phi^T \hat{P} \Phi$ form.

The alternative model-based least-squares form of policy value estimation we propose is to use the transpose of the matrix $(I - \gamma P)\Phi$ instead of Φ to minimize the residual errors in equation 3 to produce

$$\Phi^T (I - \gamma P)^T (I - \gamma P)\Phi \mathbf{w} = \Phi^T (I - \gamma P)^T \mathbf{r}. \tag{5}$$

The benefit of this approach is that it will minimize any residual errors of the vector specified by $(I - \gamma P)\Phi\mathbf{w}$ by projecting this vector onto the vector $\mathbf{r}$ [8]. The LSTD approach only minimizes the residuals if the matrix P happens to be symmetric. The reason why equation 5 is not used in LSTD is that the matrix-matrix multiplication $(I-\gamma P)^T(I-\gamma P)$ cannot be performed online. The $P \times P$ multiplication can only be performed after the probability estimates have been completely collected. For this reason, equation 5 can only be used in a model based method while equation 4 can be used in a model-free TD approach. A second improvement we have made is that we build only an $N \times k$ matrix, $\hat{P}\Phi$, instead of the full $N \times N$ matrix, $\hat{P}$. By storing a $N \times k$ matrix instead of the $k \times k$ matrix that LSTD does, we avoid the k^2 computation LSTD incurs at every sampling step in order to update its $\Phi^T \hat{P}\Phi$ matrix. We build $\hat{P}\Phi$ by

$$(\hat{P}\Phi)_{i*} \leftarrow (\hat{P}\Phi)_{i*} + \Phi_{j*} \tag{6}$$

at each transition step observation from state i to state j. Essentially, equation 6 is a transformation of a $\hat{P}_{ij}$ update to an update of the $\hat{P}\Phi$ matrix. Updating an $N \times k$ matrix costs only k additions. We therefore reduce the computation time of MBLS relative to LSTD. The trade-off is storage of an $N \times k$ matrix. But if the LSTD method stores its Φ basis vectors and since Φ is $N \times k$, then the LSTD will use $O(Nk)$ space already. Under that circumstance MBLS will improve on the computation time of LSTD without any storage increase. An important caveat to Boyan's approach is that as long as the order of states for the rows is kept the same as the order of states of the columns, then the matrix can be considered symmetric. Under this circumstance, Boyan's LSTD method will produce solutions that are as accurate as the MBLS method since both minimize residuals and both are actually forms of model-based methods.

2.1 Comparing algorithm efficiency

LSTD differs from TD by updating values of $\hat{\mathbf{w}}$ rather than the values of $\hat{\mathbf{v}}$ directly during each sampling step of a trajectory. Boyan showed that a single update of weight $\hat{\mathbf{w}}_t$ at some time t costs $O(k^2)$ where k is the number of feature/basis vectors for LSTD. Since this update must be performed at each step during sampling, then if M is the total number of sample steps taken, the complete runtime for policy value estimation of LSTD would be $O(Mk^2)$.

For our MBLS method we use a maximum likelihood (ML) approach to collect probability transition values to form the transition matrix, but instead of updating $\hat{P}_{ij} \leftarrow \hat{P}_{ij} + 1$ we build and update the $\hat{P}\Phi$ by $(\hat{P}\Phi)_{i*} \leftarrow (\hat{P}\Phi)_{i*} + \Phi_{j*}$ which costs k for each sampling step for a total of $O(Mk)$. Forming equation 5 from $\hat{P}$ and Φ and solving it costs $O(Nk^2)$ time. When using feature vectors for large MDPs, we assume $k << N$. $N \leq T$ and $T \leq M$. Therefore it is reasonable to assume $M \geq Nk$. Then the overall runtime of MBLS is $O(Mk)$. Therefore, MBLS runs faster than LSTD's $O(Mk^2)$.

Initialize $\hat{A} = \mathbf{0}$, $T = \mathbf{0}$, $\mathbf{r} = \mathbf{0}$ and set γ & basis Φ,
Repeat for each trajectory:
  Draw an initial state i according to $\boldsymbol{\pi}$
  Repeat for each step of trajectory:
    Observe next state j
    $\hat{A}_{i*} \leftarrow \hat{A}_{i*} + \phi_i$
    $T_{i1} \leftarrow T_{i1} + 1$
    $\mathbf{r}_{j1}$ = observed reward at state j
    $i \leftarrow j$
  Until state i is terminal
Repeat for each row i of T:
  $\hat{A}_{i*} \leftarrow \Phi_{i*} - \gamma \hat{A}_{i*}/T_{i1}$
Solve for $\hat{\mathbf{w}}$ in $\hat{A}^T \hat{A} \hat{\mathbf{w}} = \hat{A}^T \mathbf{r}$

Fig. 1. Model-Based Least-Squares (MBLS) policy value estimation

2.2 Comparing storage requirements

MBLS is more fault tolerant than LSTD since it does not need to maintain row and column order to minimize its residuals and it produces solutions faster. However, MBLS may use more space than LSTD if features/basis vectors are not stored. If LSTD does store feature/basis vectors, then storage is equivalent.

3 Determining basis vectors

LSTD is able to deal with large MDPs by using a linear function approximator in the form of basis vectors which compact the system into unique independent states [9]. The LSTD method requires that the feature vectors be known prior to any processing. Therefore, in problems where no prior knowledge is available, the LSTD method cannot be applied.

MBLS does not require a prior set of basis vectors. However, when no basis matrix Φ exists, the algorithm degenerates to a maximum likelihood (ML) estimation method where the probability transition matrix update becomes: $\hat{P}_{ij} \leftarrow \hat{P}_{ij} + 1$ when a sampling step transitions from state i to state j. We simply solve for $(I - \gamma \hat{P})\hat{\mathbf{v}} = \hat{\mathbf{r}}$ directly. But with the transition matrix model, it is also possible to determine a minimal spanning set of column vectors of $\hat{P} = [\hat{P}_{*1}, \hat{P}_{*2}, ..., \hat{P}_{*k}]$. This minimal spanning set forms a set of basis vectors of $\hat{P}$. By finding the basis vectors of $\hat{P}$ we can build a set of feature vectors ϕ_i from it to compress our ML model. ϕ_i is the i'th column vector of Φ composed of only zeros and ones. If $\phi_{ij} = 1$ then the j'th column of $\hat{P}$ is a member of our basis vector set for $\hat{P}$.

Figure 2 illustrates a model-based basis vector (MBBV) determination algorithm for finding basis/feature vectors of Φ of a matrix $\hat{A}$. It does so by choosing

Initialize $\Phi' = \mathbf{0}$, $\mathbf{e} = \mathbf{0}$,
Repeat for each nonzero column of $\hat{A}$:
  Let $\mathbf{a}$ be the chosen column of A at some column position i
  Choose a nonzero element in $\mathbf{a}$, call it $\hat{A}_{ij}$
  $\mathbf{e}_i = 1$
  Repeat for each nonzero column l of $\hat{A}$:
    $\hat{A}_{*l} \leftarrow \hat{A}_{il}\hat{A}_{*j} - \hat{A}_{ij}\hat{A}_{*l}$
    if$\hat{A}_{*l} = \mathbf{0}$ then
      $\mathbf{e}_l = 1$
  Append column $\mathbf{e}_l$ to Φ
(Φ on termination is a matrix composed of k $\mathbf{e}_i$ vectors $[\mathbf{e}_1\mathbf{e}_2...\mathbf{e}_k]$.)

Fig. 2. Model-based basis vector determination

a column vector $\hat{A}_{*j}$ of $\hat{A}$ and then choosing one nonzero row position i of $\hat{A}$ ($\hat{A}_{ij} \neq 0$). It then eliminates all nonzero elements in row i for all other l columns in $\hat{A}$ by: $\hat{A}_{*l} \leftarrow \hat{A}_{il}\hat{A}_{*j} - \hat{A}_{ij}\hat{A}_{*l}$. This row elimination eliminates all remaining column vectors that are linearly dependent on vector $\hat{A}_{*j}$. We then choose another vector in the remaining set of unselected column vectors and repeat the process. Since we eliminate all vectors linearly dependent on $\hat{A}_{*j}$ from our remaining set of column vectors, each iteration step increases our basis set by one linearly independent vector. Assuming a maximum of $k \leq N$ linearly independent column vectors in $\hat{A}$, MBBV will run in O(N^2k) time.

Aside from the advantage of being able to use the derived Φ feature vectors for future computation compression, the upper bound of k independent column vectors of $\hat{P}$ means that we can perform a model-based estimation by first applying ML and then using the MBBV algorithm on $\hat{P}$ to find a Φ which produces a solution in O(N^2k) time.

3.1 Comparing algorithm efficiency

Assuming no prior knowledge exists and one wished to use a temporal differencing approach, only standard TD methods and not LSTD methods may be applied to find a solution to a large MDP network. Standard TD equations run in O(MN) time where every sampling step updates all state value estimates. Without prior knowledge, MBLS degenerates to ML which takes O(M) time for data collection. With k linearly independent states, MBBV runs in O(N^2k) time. Once we have the basis vectors of Φ, we can compute $(I-\gamma\hat{P})\Phi$ in O(N^2k) time. We produce $\Phi^T(I-\gamma\hat{P})^T(I-\gamma\hat{P})\Phi$ in O(Nk^2) time to find the weight vector $\hat{\mathbf{w}}$ which can be used to derive $\hat{\mathbf{v}} = \Phi\hat{\mathbf{w}}$. The overall worst case runtime of ML with MBBV with no prior information is O(N^2k). Except in trivial systems, $M > Nk$. Therefore, ML with MBBV will run faster than TD.

3.2 Comparing storage requirements

Standard TD algorithms still require less storage, since they only store $O(N)$ state value estimates. MBLS requires the storage of the the $N \times N$ probability transition matrix. Computing the basis vectors of Φ will potentially use up $O(N^2)$ amount of memory space.

4 Finding a minimal basis set for state value estimates

```
Repeat for each nonzero of r̂:
   Find the largest nonzero r̂_i of r̂
   Find the column j of Â such at Â_ij ≠ 0
   v_i ← r̂_i/Â_ij
   Repeat for each nonzero column m of Â:
      Â_*m ← Â_im Â_*j − Â_ij Â_*m
   r̂ ← r̂_i Â_*j − Â_ij r̂
```

Fig. 3. Minimal basis search for $\hat{\mathbf{v}}$ (MBSV)

If we are only interested in finding an estimate of $\hat{\mathbf{v}}$ and do not care to collect information on Φ, then it is possible to improve our $O(N^2k)$ time to an $O(N^2l)$ time, where $l \leq k$. The basic idea is that some minimal linear combination of l independent column vectors of $\hat{P}$ will sum to the reward vector $\hat{\mathbf{r}}$: $\hat{\mathbf{r}} = \hat{P}_{1*}v_1 + ... + \hat{P}_{l*}v_l$. Let L be that minimal set of column vectors of $\hat{P}$. $l \leq k$ since there are only k independent columns in $\hat{P}$. We can find the set $\{\hat{P}_{1*}v_1 + ... + \hat{P}_{l*}v_l\}$ and the coefficients $\{v_1, ...v_l\}$ by using a method similar to MBBV, but instead of arbitrarily choosing a nonzero vector from our set of $\hat{P}$, we first find a nonzero element $\hat{\mathbf{r}}_j$ in $\hat{\mathbf{r}}$. Choose only a column vector $\hat{P}_{i*}$ where $\hat{P}_{ij} \neq 0$. Set $\hat{\mathbf{v}}_i \leftarrow \hat{\mathbf{r}}_i/\hat{P}_{ij}$. Perform row elimination as in MBBV and on the reward vector $\hat{\mathbf{r}}$. By choosing a column vector $\hat{P}_{*j}$ that has a nonzero component in i, then eliminating that component in all other column vectors, we make $\hat{P}_{*j}$ essential to the solution $\hat{\mathbf{v}}$ of $\hat{\mathbf{r}}$ since it is the only vector that can produce a nonzero value in position i. The row elimination is performed on $\hat{\mathbf{r}}$ to remove $\hat{P}_{*j}$'s effect on $\hat{\mathbf{r}}$. Figure 3 illustrates this minimal basis search for $\hat{\mathbf{v}}$ (MBSV) on a matrix $\hat{A}$. Similar to MBBV the row elimination step takes $O(N^2)$. With l independent column vectors of $\hat{P}$ which form $\hat{\mathbf{r}}$, MBSV runs in only $O(N^2l)$ time. One additional improvement made to MBSV is that we choose the largest $|\mathbf{r}_i|$ for elimination at each step in order to try and reduce the residual error $= |\hat{\mathbf{v}}_i - \hat{\mathbf{v}}|$ as early as possible so that a user may stop the iteration process early and still get a minimal residual solution.

5 Experimental Results

In these experiments, we attempt to confirm our theoretical derivations. Since none of the theoretical results depended on any specific network structure, we used randomly generated transition matrices for our MDP network. We implemented a limited linear dependence on a subset of states in order to allow for compression. The number of linearly *independent* states ranged between 10 and 100 states. All numerical results used a discount factor of $\gamma = 0.8$. All networks had at least one absorbing state which every state could reach by at least one path.

The residual errors in the tables are calculated by finding the true value estimate vector $\mathbf{v}$ by using the true transition matrix P and solving $(I - \gamma P)\mathbf{v} = \mathbf{r}$. The residual is then found by: residual $= \frac{|\hat{\mathbf{v}}-\mathbf{v}|}{|\mathbf{v}|}$. If the method being analysed generates a compressed solution $\hat{\mathbf{w}}$ of weights, then $\hat{\mathbf{v}}$ is calculated by $\hat{\mathbf{v}} = \Phi\hat{\mathbf{w}}$ and then a residual is computed. The TD algorithm updated value estimates of all states. The LSTD algorithm only produced a value estimate at termination of sampling.

Table 2. Residual and Complexity Observations

LSTD vs MBLS[1]

N	Traj	M_{LSTD}	M_{MBLS}	r_{LSTD}	r_{MBLS}	f_{LSTD}	f_{MBLS}
20	100	967	1131	0.1582	0.1490	9814	4499
50	100	2739	1990	0.2122	0.2034	25605	7067
100	200	10267	9832	0.1713	0.1667	95213	43388
1000	2000	65990	65528	0.3439	0.2829	612917	288516

TD vs ML with MBBV[1]

N	Traj	M_{TD}	M_{ML}	r_{TD}	r_{ML}	f_{TD}	f_{ML}
20	100	652	747	0.6296	0.2186	43784	20984
50	100	2766	2929	0.4232	0.1871	434362	126033
100	200	9028	8462	0.3191	0.1633	2771796	475971
1000	3000	26502	26962	0.6311	0.5075	186952190	1812695

Table 2 confirms our main assertions that both LSTD and MBLS will have the same relative accuracy assuming that the matrices are symmetric. The r_{LSTD} and r_{MBLS} columns both show comparable residual errors. When the matrix is symmetric, the main advantage of the MBLS method is a runtime savings. The floating point operations of f_{LSTD} and f_{MBLS} consistently show that MBLS requires between one half and one third the amount of processing time of LSTD.

[1] N: # of states, Traj: # of trajectories, M: # of transition steps, r: residual error, f: floating point operations, for TD $\alpha = .5$, $\lambda = .9$

Table 2 also compares the standard temporal differencing method with a ML method which then applies the MBBV algorithm to find basis compression vectors and then produces a value estimate using the compression basis vectors. Our residual computations support the general belief that model-based methods produce more accurate solutions than a model-free method such as TD. We also verify that when states can be compressed by some basis matrix Φ, the model-based method with MBBV can be competitive computationally with TD. The floating point columns of f_{TD} and f_{ML} show consistently that model-based methods with basis compression always run faster than TD when $k << N$.

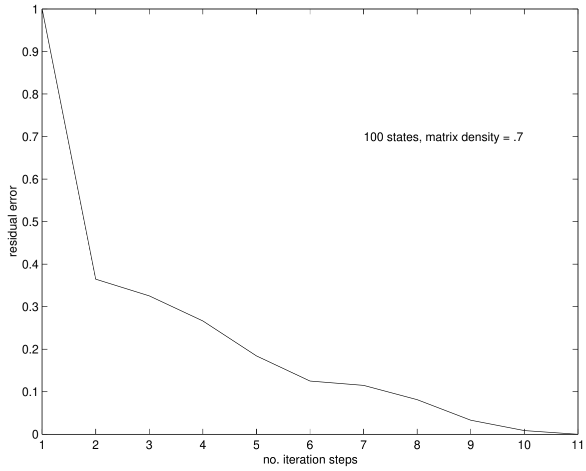

Fig. 4. residual error $= \frac{|\hat{v}-\hat{v}_i|}{|\hat{v}|}$ vs. iterations of MBSV applied to a 100x100 matrix averaging over 100 samples.

Figure 4 plots the change in normalized residual error $\frac{|\hat{v}-\hat{v}_i|}{|\hat{v}|}$ of the estimated value function $\hat{\mathbf{v}}$ over each iteration step i of the MBSV algorithm. As expected, each step provides a nontrivial improvement since each iteration improves the estimate by one independent column vector of $\hat{A}$. Figure 4 also confirms that the residuals decrease fastest in the early iterations as MBSV was designed to do.

6 Conclusions

This paper introduced an alternative policy evaluation method called model-based least-squares (MBLS), an algorithm for finding feature vectors to compress a MDP network called model-based basis vector (MBBV) determination and an iteratively improving estimator for model-based value estimation called minimal basis search for value (MBSV) estimates. The MBLS policy evaluation equation modifies the traditional least-squares temporal differencing (LSTD) equation, improving on the computation time at no extra storage cost when feature vectors

Table 3. Summary of computational differences[2]

	Prior		No prior		
	LSTD	MBLS	TD	ML w MBBV	ML w MBSV
runtime	Mk^2	Nk^2	MN	N^2k	N^2l
storage	k^2 or Nk	Nk	N	N^2	N^2
accuracy	same	same	less	more	more

are stored by LSTD. Under the assumption that a nontrivial MDP system is compressable from a set of N states to some small set of k states, but no prior feature vectors are available, then a maximum likelihood (ML) estimator with the MBBV algorithm can find the k feature/basis vectors that will compress the system and produce a policy value estimate faster than TD. If no feature/basis vectors are desired, we can run an alternate MBSV algorithm instead of MBBV which will run as fast or faster than ML with MBBV. Table 3 summaries the advantages and costs of the various methods.

In future work, we will attempt to reduce storage costs with sparse matrix methods without detrimentally affecting the improvements in accuracy and time complexity. We also plan to investigate the issue of over-fitting that can result from choosing basis vectors based on a *estimate* of the true probability transition matrix.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, Massachusetts (1998)
2. Singh, S.P., Sutton, R.S.: Reinforcement Learning with Replacing Eligibility Traces. Machine Learning **22** (1996) 123-158
3. Lu, F., Patrascu, R., Schuurmans, D.: Investigating the Maximum Likelihood alternative to TD(λ). Proceedings of the 19th ICML **1** (2002) 403-410
4. Boyan, J.A.: Least-squares Temporal Difference learning. Proc. of 16th ICML (1999) 123-158
5. Bradtke, S.J., Barto, A.G.: Linear Least-Squares Algorithms for Temporal Difference Learning. Machine Learning **22** (1996) 33-57
6. Bellman, R.E.: A Markov decision process. J. of Mathematical Mechanics **6** (1957) 679-684
7. Lagoudakis, M.G., Parr, R.: Model-Free Least Squares Policy Iteration. NIPS **14** (2001)
8. Lawson, C.L., Hanson, R.J.: Solving Least Squares Problems. Prentice-Hall, Englewood Cliffs, New Jersey (1974)
9. Koller, D., Parr, R.: Computing factored value functions for policies in structured MDPs. 16th Intl. Joint Conference on Artificial Intelligence (1999) 1332-1399

[2] Table 1 provides a list of definitions of all variables. $LSTD_{MBBV}$ & $MBLS_{MBBV}$: LSTD & MBLS with MBBV applied.