Enhanced Temporal Difference Learning Using Compiled Eligibility Traces

Peter Vamplew[1], Robert Ollington[2], and Mark Hepburn[2]

[1] School of Information Technology and Mathematical Sciences, University of Ballarat, PO Box 663, Ballarat, Victoria 3353, Australia
p.vamplew@ballarat.edu.au
[2] School of Computing, University of Tasmania, Private Bag 100, Hobart, Tasmania 7001, Australia
{Robert.Ollington, Mark.Hepburn}@utas.edu.au

Abstract. Eligibility traces have been shown to substantially improve the convergence speed of temporal difference learning algorithms, by maintaining a record of recently experienced states. This paper presents an extension of conventional eligibility traces (compiled traces) which retain additional information about the agent's experience within the environment. Empirical results show that compiled traces outperform conventional traces when applied to policy evaluation tasks using a tabular representation of the state values.

1 Introduction

Reinforcement learning addresses the problem of an agent interacting with an environment. At each step the agent observes the current state and selects an action. The action is executed and the agent receives a scalar reward. The agent has to learn a mapping from state-to-action to maximise the long-term reward. One way to do this is to learn the expected return, either per state or per state-action pair. Many algorithms for learning these values are based on the use of temporal differences (TD) [1] where the value of the current state at each step is used to update the estimated value of previous states.

It is well-established that the use of eligibility traces can substantially improve the performance of TD algorithms [2, 3]. These traces maintain a record of states and actions experienced earlier in the current episode, allowing the value of those states and actions to be updated based on the states and rewards encountered later in the episode. However conventional eligibility traces exhibit a fundamental limitation, in that they only record information about the current episode – all information about the previous episode is discarded from the traces whenever a new episode commences. (Watkins' Q(λ) algorithm [4] is even more restricted in that it clears all traces whenever a non-greedy action is performed).

In this paper we propose the use of an extended form of eligibility trace which we will refer to as a compiled trace. Compiled traces differ from conventional traces in two important ways: First, they contain information about states experienced in all previous episodes, not just those from the current episode. Second, compiled traces identify and support learning for sequences of states which are possible within the environment but which have yet to actually be observed (Note: for reasons of

A. Sattar and B.H. Kang (Eds.): AI 2006, LNAI 4304, pp. 141 – 150, 2006.

simplicity we will primarily discuss compiled traces with regards to the task of learning state values under a fixed policy (policy evaluation). However these methods can readily be extended to learning action values, as required in policy-iteration algorithms such as Q-Learning).

2 Motivation for Compiled Traces

Figure 1 illustrates a simple, probabilistic environment designed to illustrate the difference between conventional eligibility traces and compiled traces. Consider a situation in which the first two episodes observed by the agent consisted of the sequences A-C-D-F and B-C-D-G. Assuming that the estimated value of all states starts at 0, then following the first episode the values for states A, C and D will all have been trained towards the reward of -1 received on the final transition. After the second episode, the values for states B, C, and D will have been altered in the direction of the reward of +10. Notice that using conventional traces states A and B will have quite different values at this point, although it can be seen from Figure 1 that in the long run they should have identical values. This illustrates a weakness of conventional eligibility traces — by including only information gathered during a single episode they are sensitive to any stochasticity in the environment.

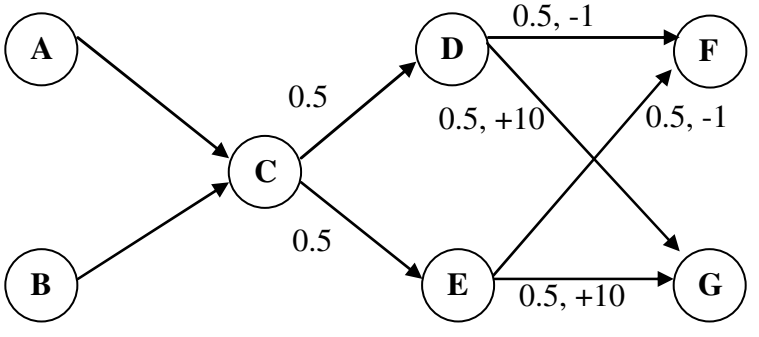


Fig. 1. An environment consisting of 7 states. A and B are the possible starting states for each episode, whilst F and G are terminal states. Arcs are labelled with their transition probabilities and associated rewards; probabilities of 1.0 and rewards of 0 have been omitted for clarity.

This issue can be addressed by extending the eligibility traces to include the values from multiple prior episodes. One relatively simple approach would be to calculate a single-episode trace as per normal, but in addition to calculate and store at each state a mean of the values of the single-episode trace at each time that state was visited. These mean-traces would then be used within the TD updates. In the previous example, this would result in state A also being updated during the second episode as state D would have 0.5 as its mean-trace value for both states A and C.

Whilst this mean-trace approach should offer some benefits compared to conventional single-episode traces in alleviating the effects of under-sampling within

probabilistic environments, it still fails to fully utilise all information available from previous episodes. Consider again the environment from Figure 1, and assume that the following three episodes have been observed: A-C-D-F, B-C-E-F and B-C-E-G. Under these circumstances the mean-trace algorithm will not update the value of state A during the third episode because the mean-trace stored for A at E is 0, as no sequence with A as a predecessor of E has ever been observed.

However from the two earlier episodes it is possible to infer that whilst the sequence A-C-E has not yet been observed, it must be possible for it to occur as the sub-sequences A-C and C-E have been observed (assuming the environment to be Markov, which is a standard assumption for many applications of TD methods). Therefore it should be possible to include state A amongst the states to be updated as a result of the third episode. A possible means by which such updates could be achieved is to use the traces stored at each state to indicate not just the extent to which other states have previously been observed as predecessors to this state (as is the case for mean-traces) but more generally the extent to which those other states could occur as predecessors to this state. We will refer to any trace algorithm which works in this manner as a compiled trace, as the trace stored at each state is essentially a compilation of all knowledge about that state's predecessors.

Construction of compiled traces can be achieved through a relatively simple bootstrapping process. Whenever a transition is observed from state s to state s′ then the compiled traces stored at s′ are updated to reflect a combination of their current values and the values of the compiled traces stored at s.

In the previous example, following the first episode state C will have a non-zero compiled trace for state A, whilst state D will have non-zero traces for both of its observed predecessors (states A and C). Following the second episode, state C will have non-zero traces for both A and B, whilst state E will have non-zero traces for both these states as well (having 'learnt' about state A via state C). Therefore when the transition from E to G occurs during the third episode, the value for state A will be updated (in addition to the values for B and C which would have been updated using conventional or mean traces).

3 Comparison to Other Learning Algorithms

3.1 Conventional Eligibility Traces

The example given in the previous section demonstrates the primary advantages of compiled traces over conventional eligibility traces. By storing information gleaned from all previous episodes rather than just the current episode, compiled traces allow a larger number of states to be updated after each step within the environment which should enable faster learning. This would be expected to be of particular benefit in environments containing a stochastic element in either the state transitions or the rewards, as under-sampling in such environments can skew the estimated values of states. By sharing updates across all possible predecessor states compiled traces reduce the effects of such stochastic variations.

3.2 Model-Based Methods

The ability to learn about sequences of states which are possible within an environment but yet to be observed can also be provided by model-based learning methods such as the Dyna methods [5]. These learning algorithms build a model of the environment by learning the successors of each state on the basis of observed transitions. This model can then be used to generate simulated episodes which can be learnt from in the same manner as episodes experienced in the actual environment.

There are some similarities between model-based methods and compiled trace methods. Both construct values reflecting the likelihood of particular sequences of states being observed on the basis of the actual sequences observed. However the model-based approach is 'forward-looking' and limited in scope in that each state learns the likelihood of every state occurring as an immediate successor, whereas compiled traces are 'backward-looking' and unlimited in scope, as each state learns about all of its predecessor states, no matter how distant.

3.3 Goal-Independent Reinforcement Learning

Some similarities also exist between compiled traces and goal-independent reinforcement learning algorithms such as DG-Learning [6] and Concurrent Q-Learning [7]. These goal-independent algorithms also update the values of states and actions which were not directly involved in the current episode. This is achieved by learning an explicit distance from every state to each other state, and using graph relaxation techniques to dynamically update these values if a shorter path is experienced.

These goal-independent algorithms have similar storage costs as compiled traces, but potentially are more computationally expensive. In addition the relaxation techniques are only applicable in the context of learning distance values, whilst compiled traces can be applied to arbitrary reward functions.

4 Implementing Compiled Traces

The main issue to be resolved in implementing compiled traces is the manner in which the current values of the traces at state s' are combined with the values stored at s when a transition from s to s' is observed (as outlined in Section 2). We believe there are three main approaches which could be utilised.

4.1 All-Visits-Mean (AVM) Traces

This algorithm sets the compiled traces at each state equal to the mean of the traces of its immediate predecessor states, weighted by the frequency of occurrence of each predecessor. Each state s in S (the set of all states in the environment) stores a compiled trace $T_{s,s'}$ for all states s' in S. Each state s also stores N_s which records the number of times that state has been visited, and $E_{s,s'}$ which sums the values of the traces of the state immediately preceding s over all episodes. All of these variables are initialised to zero. Whenever a transition from state s to s' is observed $T_{s,s}$ is

set to 1, and the values stored at s' are updated as follows (where λ is the trace decay parameter as used in conventional eligibility traces, and γ is the discounting factor):

$$N_{s'} = N_{s'} + 1 \quad (1)$$

$$E_{s',s''} = E_{s',s''} + \lambda\gamma T_{s,s''} \; \forall \; s'' \text{ in } S \quad (2)$$

$$T_{s',s''} = E_{s',s''} / N_{s'} \; \forall \; s'' \text{ in } S \quad (3)$$

If state s' is the first state in an episode, then step 2 is omitted (effectively combining the current compiled trace with an all-zero predecessor trace).

AVM traces reflect the best approximation to the predecessor probabilities observed so far, and so should promote rapid, accurate learning. However they weight all episodes equally regardless of their currency, which means the traces will adapt slowly (and never fully) to dynamic environments in which the transition probabilities change over time. This is a significant issue for the use of compiled traces for policy iteration, as in Q-learning [4]. Even if the environment itself is static, the policy followed will change over time, so retaining information from previous episodes may result in traces which no longer reflect the current policy. This is an issue with any compiled trace algorithm, but particularly for the AVM form due to its infinitely long memory of previous episodes.

4.2 Recent-Visit-Mean (RVM) Traces

To adapt to dynamic environments, the traces must forget about episodes which are no longer relevant. One means to achieve this is to calculate the mean of the predecessor traces over a fixed number of recent visits to this state. This can be implemented using a queue at each state to store the predecessor traces for the most recent visits. The sensitivity of the agent to environmental changes can be altered by varying the length of the queue (which we will denote as v). Smaller values for v will place a greater emphasis on more recent learning episodes.

4.3 Blended Traces

This approach aims to be better suited to dynamic tasks than AVM traces by placing a larger weighting on more recent experience. In this algorithm the traces at s' are updated following a transition from s, as follows (where β is a blending parameter with $0 \leq \beta \leq 1$):

$$T_{s',s''} = \beta T_{s',s''} + (1 - \beta)\lambda\gamma T_{s,s''} \; \forall \; s'' \text{ in } S \quad (4)$$

If s' is an initial state then the update is:

$$T_{s',s''} = \beta T_{s',s''} \; \forall \; s'' \text{ in } S \quad (5)$$

The emphasis which the system places on recent experience is controlled via the β parameter – larger values of β will place more weight on accumulated past experience (as stored in $T_{s',s''}$), whilst smaller values will make the system more responsive to recent experience ($T_{s,s''}$).

One possible problem with using a static value for β is that it will take several visits to a state before the traces grow significantly from their initial zero values,

which will slow early learning. A potential solution is suggested by viewing AVM traces as a specialised form of a blending trace where the value of β is determined dynamically at each state to be equal to $(N_{s'} - 1)/N_{s'}$. It may be possible to combine the early learning speed of this approach with the on-going flexibility of blending by setting $\beta = ((N_{s'})^n-1)/ (N_{s'})^n$ where n is a fixed value just smaller than 1, or alternatively $\beta = \min((N_{s'} - 1)/N_{s'}, \beta_{max})$ where β_{max} is an upper-bound on the value of β. In this paper we have used the latter approach.

4.4 Comparison of Compiled Trace Algorithms and TD(λ)

Both RVM and blended traces can be seen as generalizations of the AVM form of compiled traces. Behaviour identical to AVM can be produced by setting the algorithm's parameter appropriately (using queue length $v=\infty$ for RVM or $\beta_{max}=1$ for blended traces). Similarly both algorithms are a generalization of conventional TD(λ), which can be obtained by using $v=1$ or $\beta_{max}=0$. Therefore previous proofs of the convergence properties of TD(λ) (such as [8]) apply to these compiled trace algorithms, at least for these specific choices of parameter. Extension of these proofs to the more general form of these algorithms is an area for future work.

5 Experimental Method

5.1 The Layered-DAG Test Suite

The experiments reported in this paper are based on a suite of policy evaluation test-beds (the Layered-DAG tests). These test environments are defined by a directed acyclic graph, in which each node represents a state. The nodes have been grouped into layers with arcs from each node in a layer to each node in the next layer. Each arc is assigned a random probability of occurrence (with the probabilities on the outgoing arcs from each node normalised so as to sum to 1). Each episode commences at a randomly selected node in the first layer, and ends when the environment moves to a terminal state in the final layer. Each terminal node is assigned a unique fixed reward in the range 0..n-1, where n is the number of nodes in the terminal layer.

This test suite has two advantages for this research. First the true value of each state in the environment can readily be calculated. Secondly a range of problems of varying difficulty can be generated by altering the number of layers in the graph or the number of nodes in each layer. This allows an investigation of the relative ability of compiled and conventional traces as the problem difficulty is increased.

For each variant of the Layered-DAG problem, various combinations of parameters were trialed for both the conventional and compiled traces. For each set of parameters, 20 trials were run using different random number generators to assign the transition probabilities within the graph. Each trial consisted of a number of episodes equal to 50 multiplied by the number of nodes per layer, to ensure that on average each state would experience the same number of visits regardless of the size of the test problem.

After each learning episode, the root mean squared error of the table values compared to the actual state values was calculated. This was averaged over all

episodes of each trial, and then over each trial for a given set of parameters. This measure was chosen as it requires the learning algorithm to converge both rapidly and to a good approximation of the actual values — both of these properties are important if the algorithm is to be applied in an on-line context.

5.2 Choice of Compiled Trace Algorithm

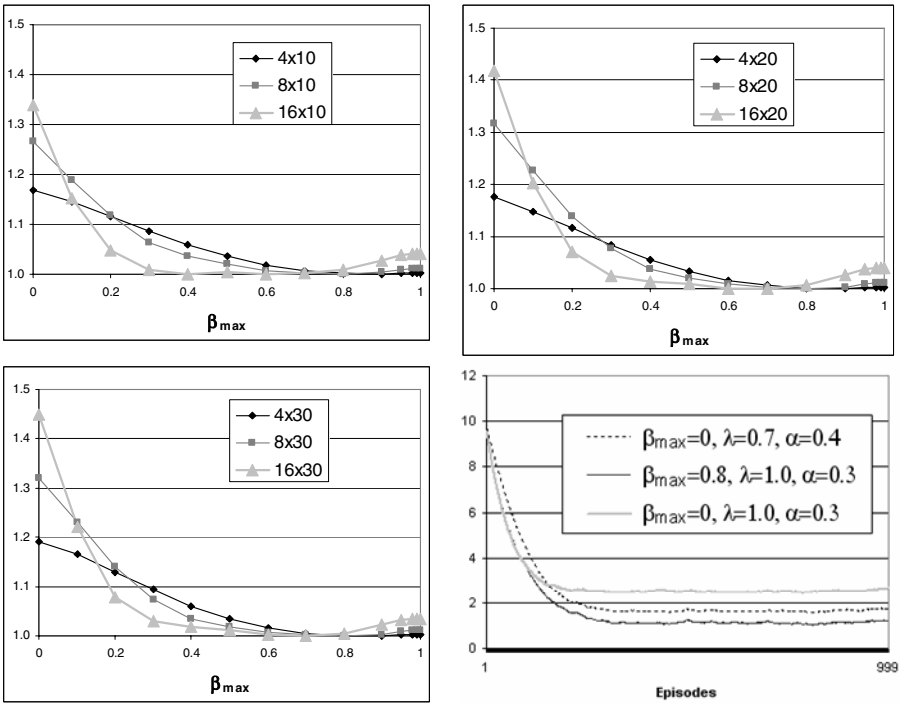
Both the RVM and blended forms of the compiled trace algorithm were investigated. Very little variation was noted between the two algorithms, and hence for space reasons only the blended trace results will be reported in this paper. Given the similar performance of these two variants of compiled traces the blended form is to be preferred due to the much greater space requirements of the RVM form.

6 Results and Discussion

The graphs in Figures 2–4 summarise the performance of the blended trace algorithm across all variants of the Layered-DAG problem (each variant is identified as NxM where N is the number of nodes per layer, and M is the number of layers). The results have been normalized relative to the lowest mean RMSE error obtained on each variant of the problem, to facilitate comparison across these variants. Note that the first data-point in each graph (corresponding to $\beta_{max}=0$) indicates the performance of conventional TD(λ).

It can immediately be seen that non-zero values of β_{max} significantly improve on the results achieved using standard TD(λ). The TD(λ) error rate is on the order of 15-45% higher than that obtained using the optimal value of β_{max}. Importantly the trend is that the improvement due to using compiled traces increases as the complexity of the problem (either the number of states per layer or the number of layers) is increased.
The performance of the compiled trace algorithm is relatively insensitive to the choice of β_{max}, with good results obtained across a wide range of values. As the episode length increases, the optimal choice of β_{max} gradually decreases.

An examination of the best values found for the α (learning rate) and λ parameters gives some insight into the reasons for the improved performance achieved using compiled traces. Whilst the α values were similar for both algorithms, the best results for compiled traces were achieved using far higher values for λ than were beneficial for TD(λ). Figure 5 illustrates this for the 8x20 Layered-DAG problem, showing the mean RMSE curves achieved during learning for both algorithms using their optimal parameter settings and for TD(λ) using the compiled trace algorithm's optimal settings. A higher λ value produces rapid early learning when used in conjunction with either conventional or compiled traces, by maintaining higher traces for (and therefore making larger changes to the values of) states encountered early in each episode. However TD(λ) fails to converge to a suitably low final error when using this λ value, as these large trace values cause the values of early states to be overly reactive to the reward received in the most recent episode. In contrast using compiled traces, the change due to most recent reward are 'spread' across many possible predecessor states, thereby moderating the impact of the stochasticity in the environment.



Figs. 2–5. Graphs of the normalised root-mean-squared-error (averaged over all episodes of all trials) against the β_{max} parameter for the 10 layer (top-left), 20 layer (top-right) and 30 layer (bottom-left) variants of the Layered-DAG task. The bottom-right graph plots RMSE (averaged over all trials) against learning episodes on the 8x20 task.

7 Conclusion and Future Work

7.1 Comparison of Conventional and Compiled Traces

This paper has demonstrated both by example and through empirical results that the use of compiled eligibility traces can substantially improve the performance of an agent using temporal difference learning methods, particularly in highly probabilistic environments. On the Layered-DAG tasks compiled traces gave a 15-45% reduction in error compared to conventional traces (implemented by setting the β_{max} parameter to 0).

Thus far these benefits have been demonstrated only in the simplest of TD learning tasks (evaluation of a static policy using a tabular representation of the state values). Further experiments will be required to determine the effectiveness of compiled traces on more difficult problems, particularly those involving dynamic environments or policy iteration tasks. It would also be beneficial to formally analyse the convergence properties of TD algorithms using compiled traces.

The primary disadvantage of compiled traces compared to conventional traces is the additional memory and computational overhead — a naïve implementation of

TD(λ) is order O(|S|), whereas a similarly naïve compiled traces implementation is $O(|S|^2)$. It has been shown that the costs of TD(λ) can be reduced by ignoring near-zero traces — this technique may be less applicable to compiled traces however as the proportion of near-zero traces is likely to be significantly lower. However it may still be possible to discover more efficient implementation techniques similar to those used for TD(λ) by [9].

A further issue which needs to be addressed in order to scale up compiled traces to larger problems is integrating it into function approximation algorithms, as has previously been achieved for conventional traces. We intend to explore the use of compiled traces with localized function approximators such as tile-partitioning [10] and resource-allocating networks [11].

7.2 Using Compiled Traces for Planning and Exploration

Whilst the primary motivation behind compiled traces is to maximise the amount of learning possible from each interaction with the environment, it may also prove possible to utilise the additional information stored in these traces for other purposes. One possible application would be within an intelligent exploration algorithm. Such an algorithm might identify a particular state worthy of further exploration (for example a region of state space which has been under-explored, or not visited recently) and set it as a sub-goal. With conventional eligibility traces there is no knowledge within the system of how to move from the current state towards this sub-goal state, and hence no way to select actions to carry out the desired exploration.

Compiled traces also do not directly encode this information in the way that a goal-independent learning system does. However the values of the traces at the sub-goal state may give some guidance as to the correct actions. If the system is currently at state *s* and the sub-goal is at state *g*, we would select the action *a* which gives the highest value for $T_{g,s,a}$ (the trace stored at g for the combination of s and a). If $T_{g,s,a}$ is zero for all values of *a*, then we could either select an action randomly, or select greedily with respect to the overall goal. This process would be repeated for all states encountered until state *g* is reached. Whilst this approach is unlikely to result in the optimal path from *s* to *g*, it is likely to be significantly more efficient than random exploration in search of *g*, and incurs a relatively small computational cost.

This may be an area in which compiled traces provide benefits relative to model-based approaches, as the 'backward-looking' nature of compiled traces may allow more efficient planning of exploration than the 'forward-looking' models can achieve.

References

1. Sutton, R.S. (1988). Learning to predict by the methods of temporal differences, Machine Learning, 3:9-44.
2. Singh, S.P. and Sutton, R.S. (1996), Reinforcement learning with replacing eligibility traces, Machine Learning, 22:123-158.
3. Rummery, G. and M. Niranjan (1994). On-line Q-Learning Using Connectionist Systems. Cambridge, Cambridge University Engineering Department.
4. Watkins, C.J.C.H. (1989), Learning from Delayed Rewards, PhD Thesis, Cambridge University.

5. Sutton, R.S, (1991), Dyna, an integrated architecture for learning, planning and reacting. SIGART Bulletin, 2:160-163.
6. Kaelbling, L. P. (1993). Learning to Achieve Goals, Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, Chambéry, France.
7. Ollington, R. and Vamplew, P. (2003), Concurrent Q-Learning for Autonomous Mapping and Navigation, The 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems, Singapore.
8. Jaakkola, T., Jordan, M.I. and Singh, S.P. (1994), On the convergence of stochastic iterative dynamic programming algorithms, Neural Computation, 6:1185-1201
9. Cichosz, P. (1995), Truncating temporal differences: On the efficient implementation of TD(λ) for reinforcement learning, Journal of Artificial Intelligence Research, 2:287-318.
10. 10.Sutton R.S. (1996). Generalisation in reinforcement learning: Successful examples using sparse coarse coding. In Touretzky D.S., Mozer M.C., & Hasselmo M.E. (Eds.). Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference (1038-1044). Cambridge, MA: The MIT Press.
11. Kretchmar, R. M. and Anderson, C.W. (1997). Comparison of CMACs and RBFs for local function approximators in reinforcement learning, IEEE International Conference on Neural Networks.