

Reducing the Time Complexity of Goal-Independent Reinforcement Learning

Robert Ollington
 School of Computing
 University of Tasmania
 Sandy Bay, Tasmania, Australia
 Robert.Ollington@utas.edu.au

Peter Vamplew
 School of Computing
 University of Tasmania
 Sandy Bay, Tasmania, Australia
 Peter.Vamplew@utas.edu.au

ABSTRACT

Concurrent Q-Learning (CQL) is a goal independent reinforcement learning technique that learns the action values to all states simultaneously. These action values may then be used in a similar way to eligibility traces to allow many action values to be updated at each time step. CQL learns faster than conventional Q-learning techniques with the added benefit of being able to apply all experiences gained performing one task to any new task within the problem domain. Unfortunately the update time complexity of CQL is $O(|S|^2 \times |A|)$. This paper presents a technique for reducing the update complexity of CQL to $O(|A|)$ with little impact on performance.

1. INTRODUCTION

Concurrent Q-Learning (CQL) [1, 2] is a novel form of reinforcement learning, based on Watkin's $Q(\lambda)$ [3, 4] that learns action values to all states simultaneously, irrespective of which state is the current goal. Since this form of learning is goal-independent, action values learned in one task may be applied immediately to a new task. CQL is similar to DG-learning [5], except that the extra inter-state information learned is used to allow many more action values to be updated at each time step. This means that, even for a fixed goal, CQL learns faster than Q-learning. Unfortunately both CQL and DG-learning have worst case time complexity of $O(|S|^2 \times |A|)$.

Hierarchical reinforcement learning is a developing field of RL that attempts to divide a task into a hierarchy of subtasks. [6] develops a hierarchical form of the DYNA architecture [7] that improves the scalability of the technique and facilitates the transfer of learning between tasks. [8] explores a hierarchical version of the DG-learning algorithm that considerably reduces the complexity of both learning and action selection with little loss in path quality. Other approaches include the task-based hierarchy of [9] and more recently, the MAXQ algorithm [10], which was shown to learn significantly faster than Q-learning, under certain conditions.

The current work investigates a hierarchical version of CQL where states are given a randomly allocated training threshold. Only those states that are within a certain range, dictated by the training threshold, of the current state have their action values updated. It is shown that, for randomly allocated thresholds, the worst case time complexity of the algorithm is $O(|A|)$. Preliminary results suggest that, in accordance with other hierarchical methods, the impact on path quality is

minimal. The key advantages of the new algorithm include a dramatic improvement in update time and memory requirements, automatic establishment of a state hierarchy, and goal independence.

2. CONCURRENT Q-LEARNING

Q-Learning [3, 4] is a form of reinforcement learning (RL) that utilises the temporal difference (TD) learning rule [11]. The TD learning rule updates the value of a state at one time step by considering the value of the state reached in the succeeding time step according to equation (1).

$$\delta \leftarrow r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \quad (1)$$

δ is the error in the value of the state occupied at time t , $V(s_t)$. r_t is the reward experienced at time t , and γ is a discounting factor that determines the extent to which future rewards are considered. This learning rule updates the value of a state with respect to the policy currently being followed and is called an on-policy method.

The rule may be extended to action values by replacing the state values $V(s)$ with the corresponding action values $Q(s, a)$. $Q(s, a)$ is the predicted discounted future reward for performing action a from state s . Sarsa [12], an algorithm using the action value learning rule, is also an on-policy method since the update is dependent upon the action chosen at state s_{t+1} .

```

Initialise  $Q(s, a); e(s, a) \forall s, a$ ,
Initialise  $s, a$ 
Repeat:
  Take action  $a$ ; observe  $r, s'$ 
  Choose  $a'$  from  $s'$  using policy derived from  $Q$ 
   $a^* \leftarrow \arg \max_b Q(s', b)$ 
   $\delta \leftarrow r + \gamma Q(s', a^*) - Q(s, a)$ 
   $e(s, a) \leftarrow e(s, a) + \delta$ 
  for all  $s, a$ :
     $Q(s, a) \leftarrow Q(s, a) + \alpha \delta e(s, a)$ 
    if  $a' = a^*$ 
       $e(s, a) \leftarrow \gamma \lambda e(s, a)$ 
    else
       $e(s, a) \leftarrow 0$ 
   $s \leftarrow s'; a \leftarrow a'$ 
    
```

Figure 1: The Watkin's $Q(\lambda)$ algorithm. α is the learning rate and $e(s, a)$ is the eligibility trace. Note that eligibility traces must be reset when a non-optimal action is chosen. [13]

Q-learning is an off-policy variant of TD-learning. The error in the action value $Q(s_t, a_t)$ is calculated from the reward and the *best* action value from the subsequent

state. As is the case with other TD-learning algorithms, Q -learning may be further enhanced by maintaining a trace of previously performed actions and using this eligibility trace [11, 14] to update the values of those action as well as the most recently performed action. Watkin's $Q(\lambda)$ [3, 4] is one way to implement eligibility traces and the algorithm is given in Figure 1.

While Q -learning is an off-policy method, it is unfortunately goal-dependent since the reward term is based on the goals of the current task. When the task changes, invariably the reward structure changes also. This invalidates previously learned action values, which may cause interference as the agent tries to learn the new task.

Concurrent Q -Learning (CQL) [1, 2] solves this problem by simultaneously learning the action values for reaching any state from any other state. Given an estimate of the expected reward at each state, the algorithm chooses a goal, and uses the learned action values for that state to choose an action. The CQL algorithm may be implemented using eligibility traces, as in Watkin's $Q(\lambda)$, however the extra information obtained by CQL may be used to replace the traces by the corresponding action values [2]. Additionally, the relaxation procedure presented in [5] is used to check action values for consistency prior to action selection as shown in Figure 2.

Initialise $Q^d(s, a) \forall s^d, s \in S, \forall a \in A$
 Initialise s, a
 Repeat:
 Take action a ; observe s'
 Relaxation
 For each state (destination), $s^d \in S$:
 For each action a' and intermediate state, $s' \in S$:
 if $Q^s(s', a') \times \max_a Q^d(s', a) > Q^d(s', a')$
 $Q^d(s', a') \leftarrow Q^s(s', a') \times \max_a Q^d(s', a)$
 Choose a' from s'
 For each state (destination), $s^d \in S$:
 $\delta \leftarrow \gamma \max_b Q(s', b) - Q^d(s, a)$
 For all state-action pairs (s^o, a^o) :
 if $Q^s(s^o, a^o) \times Q^d(s, a) \geq Q^d(s^o, a^o)$
 $Q^d(s^o, a^o) \leftarrow Q^s(s^o, a^o) [Q^d(s, a) + \alpha \delta]$
 $s \leftarrow s'; a \leftarrow a'$

Figure 2: The CQL algorithm. $Q^d(s, a)$ is the action values for reaching state s^d from s via action a .

The criteria for a Q -value $Q^d(s^o, a^o)$ to be updated is that the action just performed must be on the shortest path from the state s^o , via action a^o to the destination s^d . A key advantage of CQL over similar methods, such as DG-learning [4], is that many more action values may be updated from the same experience.

This is clearly demonstrated by the example shown in Figure 3. An agent familiar with the environment in Figure 3 has just moved along the path A-B-C-D-E and attempts to move to F. However, the doorway, previously open, has been blocked. Clearly, the value of taking the action E→F, with respect to the goal G, should be reduced. CQL and DG-learning correctly

make this update. However, any action for which the optimal path to G previously included the action E→F should also have its value reduced. The actions for which this is the case are identified in bold. DG-learning will not update these values since relaxation can only find shorter route. If eligibility traces are included, CQL will correctly update action B-C, C-D and D-E; and if the full CQL algorithm is used, as shown in Figure 2, all possible updates, as indicated in Figure 3, will be made.

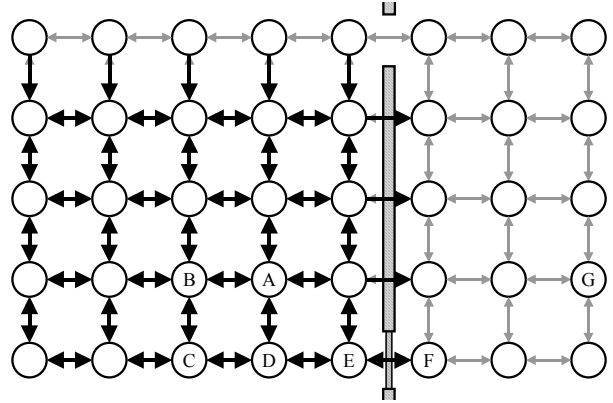


Figure 3: A navigational problem consisting of a grid of states with possible actions to each adjacent state. A wall with two 'doorways' divides the environment into two regions.

CQL is completely goal independent, which means that it is able to apply experiences gained in one task to the next without any interference, a very desirable property for any learning algorithm. Furthermore, due to the additional updates made, CQL learns faster than Q -learning even in goal-directed tasks. This is particularly evident when the environment changes, CQL adapts to changes quickly and intelligently to find detours and shortcuts [2].

3. REDUCING THE DIMENSIONALITY

While Q -learning and, by extension, CQL are efficient leaning algorithms in terms of the number of time-steps taken to learn optimal solutions [15], they both suffer the 'curse of dimensionality' with respect to the update time per step. Given a state space S and action space A , the worst case update time complexity for Watkin's $Q(\lambda)$ is $O(|S| \times |A|)$, while for CQL it is $O(|S|^2 \times |A|)$. Lazy learning may be applied to $Q(\lambda)$ to reduce the complexity to $O(|A|)$ as in the "Fast Online $Q(\lambda)$ " algorithm [16], however such techniques would be more difficult to apply to CQL, and at best would reduce the complexity to $O(|S| \times |A|)$.

One common approach to this type of problem is to employ a tree data structure. [17] found that human subjects organised spatial landmarks in a hierarchical manner, and it seems likely that other information may also benefit from being organised in this way. If a tree of states were used in conjunction with CQL then each state would need to learn action values for each of its siblings, each of its parent's siblings, each of its parent's parent's siblings and so on, as shown in Figure 4. Additionally, all of a state's siblings would have their action values updated whenever an action is performed from that state.

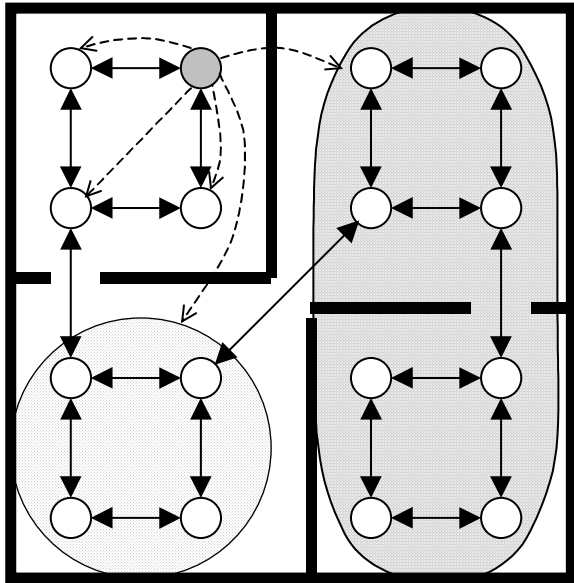


Figure 4: Top: A group of states that may be best represented as hierarchical groups. Solid arrows show possible actions, dotted lines show action values that would need to be learned from the shaded state. The shaded ovals show some conceptual groups. Bottom: A tree structure representing the environment on the left. The leaf nodes of the tree represent the states themselves; other nodes represent a conceptual grouping of the states. Dotted arrows show the action values, corresponding to the diagram on the left, that must be learned from the shaded state. Shaded circles correspond to the conceptual groups in the top diagram.

Previous work has considered similar hierarchical structures to reduce the complexity of reinforcement learning algorithms [6, 8-10, 18]. For the CQL algorithm operating on $|S|$ states arranged in a balanced tree structure of degree d , this approach would yield the theoretical worst case update time shown in equation (2).

$$W(|S|) = [(d-1) \times \log_d |S|]^2 = O(\log |S|) \quad (2)$$

While this is a significant improvement, there are likely to be few real world problems for which a tree structure can be determined prior to training. For all other problems, the tree structure would need to be determined dynamically. While [19] presents an algorithm that learns such a structure dynamically, it is not clear that such a technique could easily be applied to CQL.

3.1. TRUNCATED CQL

An alternate, but similar, approach would be to identify some states as being more important than others. These key states would be similar to the parent nodes in the tree structure with states learning action values to other states based on their proximity and the degree of importance placed upon them, and ignoring states of lesser importance. An algorithm for this truncated form of CQL is given in Figure 5.

```

Initialise  $Q^d(s, a) \forall s^d, s \in S, \forall a \in A$ ,
Initialise  $s, a$ 
Repeat:
  Take action  $a$ ; observe  $s'$ 
  Choose  $a'$  from  $s'$ 
  For each state  $s^d \in S \mid Q^d(s', a') > T(s^d)$ :
     $\delta \leftarrow \gamma \max_b Q^d(s', b) - Q^d(s, a)$ 
    For each  $s^o \in S \mid \max_b Q^d(s^o, b) > T(s)$ :
      For each  $a^o \in A$ 
        if  $Q^o(s^o, a^o) \times Q^d(s, a) \geq Q^d(s^o, a^o)$ 
           $Q^o(s^o, a^o) \leftarrow Q^o(s^o, a^o) [Q^d(s, a) + \alpha \delta]$ 
     $s \leftarrow s'; a \leftarrow a'$ 

```

Figure 5: The truncated CQL algorithm (T-CQL). $T(s)$ is the training threshold assigned to state s . Low thresholds can be considered to represent high importance or key states.

Note that the value update algorithm for T-CQL omits the triangular inequality updates of CQL. It was found that these updates were not necessarily valid in the truncated algorithm. The role of these updates, primarily one of finding shorter paths, has been transferred to the action selection algorithm, which is discussed below.

The update procedure in Figure 5 is not sufficient to solve the problem since states with high thresholds (low importance) will never learn action values to distant states with similarly high thresholds. In order to choose an action that will lead to such a state the agent needs to search for an intermediate key state with a low threshold that has legitimate action values for the target. The agent then begins to move in the general direction of the target by first moving towards the closest of these intermediate states. As it moves towards the intermediate states, other states in closer proximity to the target may become known and the trajectory changes towards this state, as shown in Figure 6.

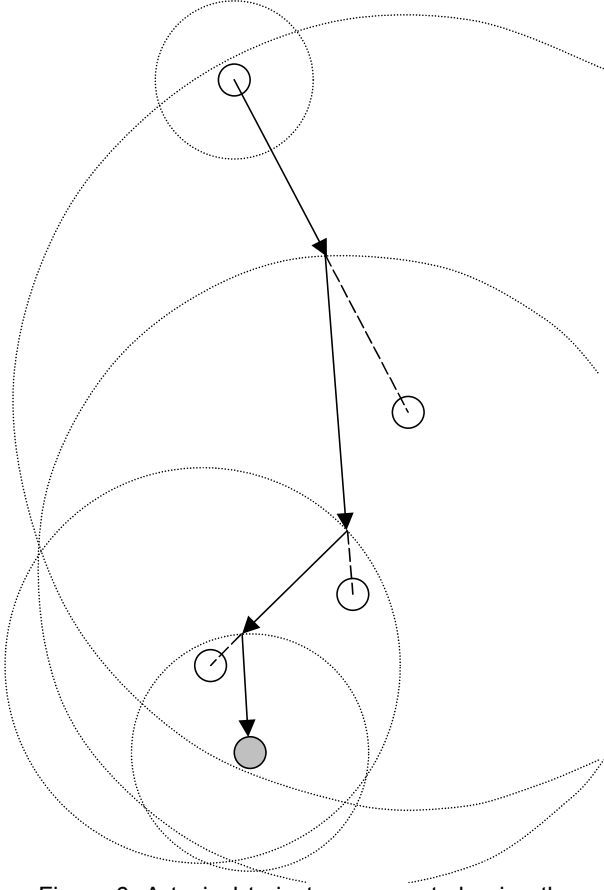


Figure 6: A typical trajectory generated using the T-CQL algorithm. Solid circles represent states; corresponding dotted circles represent their training thresholds. Dashed lines show the planned path; solid arrows show the actual path taken.

While Figure 6 shows that non-optimal paths may be generated, in practice optimal paths are often found, either through the use of redundancy in choosing key states, or simply because there are a finite number of actions that may be chosen from any state. The action selection algorithm, as depicted in Figure 6, is given below in Figure 7.

$$\begin{aligned}
 s^i &\leftarrow \arg \max_x \left[\max_b Q^x(s, b) \times \max_b Q^y(x, b) \right] \\
 &\quad | x \in S, \max_b Q^x(s, b) > T(x) \\
 a &\leftarrow \arg \max_b Q^i(s, b)
 \end{aligned}$$

Figure 7: The action selection algorithm for the T-CQL algorithm. s^i is the target state, a is the action that will be performed from the current state, s .

This action selection strategy will be sufficient, provided the agent has explored enough to find suitable key states with low thresholds. In the early stages of training, this may not be the case and problems may arise. For example, the agent may reach a key state with accurate action values to the current target; the subsequent state may not have learned about the target and so it searches for a suitable intermediate state. At this point, the action values of the previous key-state would not normally be updated because the current state has no information about the target. If little exploration has been

undertaken at this point, the agent may find that the most suitable intermediate state is the key state just visited. The agent will then return to the key state and continue back and forth between the two states indefinitely.

To solve this problem, action values are updated for each state that meets the threshold criterion, and for the current target. This may lead to a certain degree of forgetting in the early stages of training since, as in the example above, a key state may have its action values erroneously updated based on incorrect information from the subsequent state. The advantage, however, is that it encourages exploration by forcing the agent to choose an alternate route from the key state.

The final issue is the choice of thresholds or key states. While this may be easier than finding a tree structure for the states, it may still be difficult or impossible to identify key states prior to training. However, it was found that, provided a reasonably conservative distribution function was chosen, the thresholds could be assigned randomly. In keeping with the tree-like nature of the algorithm, thresholds were chosen from an exponential distribution as shown in equation (3).

$$f(x) = \frac{ae^{ax}}{e^a - 1} \quad (3)$$

3.2. UPDATE COMPLEXITY

The worst case time complexity for both the update and action selection algorithms will occur when training is near completion since action values start at zero and more updates are performed when more action values are higher than thresholds. It will also occur for a state near the conceptual centre of the environment, since this state within the threshold of a larger number of states than a state at the edge of the environment.

To derive an expression for the update time complexity will consider a simple environment consisting of states arranged in a two-dimensional plane. Each state has neighbours to the north, south, east and west, with no barriers. The number of states that are r steps from the central state is $4r$. Therefore the number of states, $N(r)$, which are within R units of the central state, and which need to be considered in the update algorithm is:

$$N(r) = \sum_{r=1}^R 4r \mathcal{P}(T < \gamma^r) \quad (4)$$

For any threshold probability distribution that it not asymptotic at $r=0$, this is a convergent series. For example, if the thresholds are distributed evenly between 0 and 1:

$$N(r) = \sum_{r=1}^R 4r \gamma^r = \frac{4\gamma}{(1-\gamma)^2} \quad (5)$$

This may easily be extended to the general case giving worst case time complexities for the update and action selection algorithms of $O(|A|)$.

4. RESULTS

The T-CQL algorithm was tested in a complex office-like environment consisting of 256 states as shown in Figure 8. Possible actions consisted of moving north, south, east or west. The agent was required to navigate between successive pseudo-random locations within the environment. The successful traversal from one location to another constituted one episode.

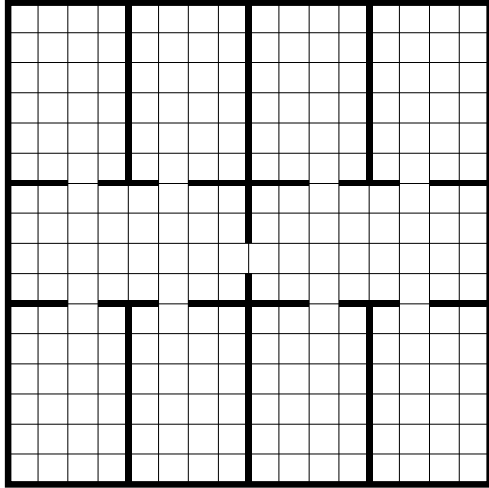


Figure 8: The environment used for testing T-CQL. Thick lines represent walls; thin lines represent state divisions.

Threshold values were chosen from random exponential distributions as shown in equation 5. These results were compared with those for the full CQL algorithm with all thresholds equal to zero. Several threshold distributions were considered with the parameter a , in equation 5, taking values 0 (flat), 2, 4, and 6. The performance is shown in Figure 9.

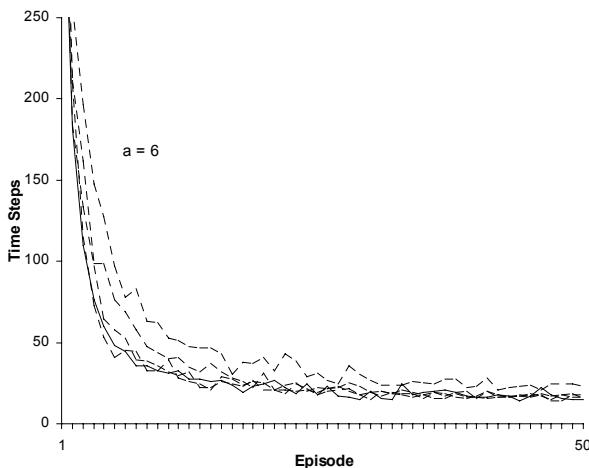


Figure 9: Performance comparison of T-CQL and CQL. The solid line shows the performance of the full CQL algorithm with dotted lines showing the performance of T-CQL with thresholds chosen randomly from exponential distributions with the parameter a as shown.

Figure 9 shows that the final performance of all but the most extreme ($a=6$) threshold selections was comparable to the full version of CQL. The learning rate for $a=0$ and $a=2$ appear comparable to the full version, with a slight decline as a increases further.

Figure 10 shows the average number of updates required per time step for the same set of threshold distributions. T-CQL performed approximately $1/8^{\text{th}}$, $1/25^{\text{th}}$, $1/60^{\text{th}}$ and $1/150^{\text{th}}$ the number of updates compared to CQL for values of a equal to 0, 2, 4 and 6 respectively. Of all threshold distributions tested, only $a=6$ came close to reaching its theoretical maximum number of updates in this environment.

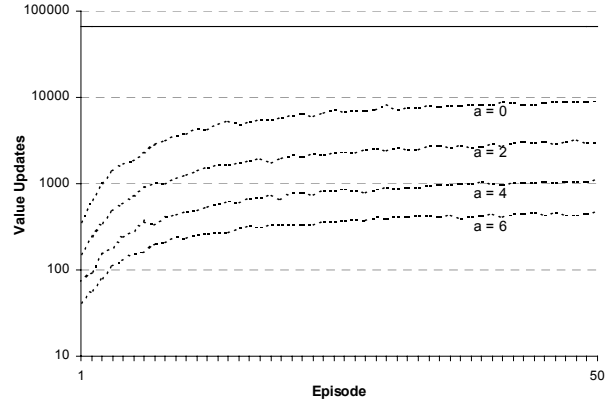


Figure 10: The average number of updates made per time step for each episode. The solid line is for the full version of CQL; dotted lines show the values for T-CQL for the given threshold distributions.

In order to get an indication of how well the performance of T-CQL scales as the number of states increases the observed path length was also compared to the optimal path length for each threshold distribution. The results are shown in Figure 11 and demonstrate that, for conservative threshold distributions, T-CQL should scale well as the number of states, and hence average path length, increases. However, for an exponential threshold distribution with $a=6$, the performance degraded rapidly as the goal distance increased.

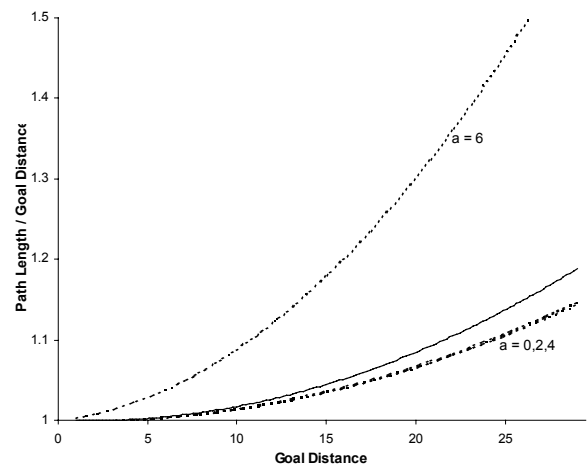


Figure 11: T-CQL scaling as goal distance increases. The solid line shows the trend line for the complete version of CQL; dotted lines show trends for T-CQL. Trend lines for a equal to 0, 2 and 4 are indistinguishable.

5. CONCLUSION

CQL is a goal independent reinforcement learning algorithm that learns faster than conventional Q-learning even in goal directed tasks. Unfortunately the update

time complexity is very poor at $O(|S|^2 \times |A|)$, making the algorithm unsuitable for large tasks involving many states. The T-CQL algorithm presented here offers a dramatic improvement in update time complexity with little or no loss in performance of CQL. T-CQL updates in worst case time complexity $O(|A|)$.

T-CQL achieves this improvement by updating selected action values only based on the proximity of target states and on a training threshold that varies from state to state. With appropriate selection of thresholds, a hierarchy of states is established. Action selection then consists of locating a state with updated action values to the goal, and for which there are also updated action values from the current location to that intermediate state. As the agent moves towards this intermediate state new information becomes available, eventually allowing direct movement towards the goal.

It was shown that a random selection of thresholds is sufficient to solve the task and that the number of action values updated converges as the size of the state space increases, provided the distribution of thresholds is not asymptotic at zero. Since memory requirements are also reduced, this effectively removes the size of the state space as a constraint on the use of reinforcement learning and in particular CQL.

While initial investigations suggest that, in most cases, the performance of T-CQL scales as well, if not better than, CQL as the number of states increases, further work needs to be conducted to confirm this. For threshold choices showing poor performance, it is possible that more look-ahead steps may be a solution. The current implementation chooses actions by looking for at most one intermediate state. By increasing the number of intermediate steps considered to some fixed number, performance may be improved with no significant impact on the time complexity of the algorithm. The current study considered only the case where thresholds were chosen randomly and were static throughout training. In future work various methods of dynamically adjusting thresholds will be considered to improve the time performance of the algorithm and to reduce training time.

REFERENCES

- [1] R. B. Ollington and P. W. Vamplew, Concurrent q-learning for autonomous mapping and navigation, *Proceedings of the Second International Conference on Computational Intelligence, Robotics and Autonomous Systems*, Singapore, 2003
- [2] R. B. Ollington and P. W. Vamplew, Concurrent q-learning: Goal independent reinforcement learning, *Neural Computation*, Vol. pp. 2004
- [3] C. J. C. H. Watkins, Learning from delayed rewards, 1989
- [4] C. J. C. H. Watkins and P. Dayan, Q-learning, *Machine Learning*, Vol. 8, pp. 279-292, 1992
- [5] L. P. Kaelbling, Learning to achieve goals, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambéry, France, 1993
- [6] S. P. Singh, Reinforcement learning with a hierarchy of abstract models, *Proceedings of the National Conference on Artificial Intelligence*, 1992
- [7] R. S. Sutton, Dyna, an integrated architecture for learning, planning and reacting, In *Working notes of the {aaai} spring symposium on integrated intelligent architectures* ed), pp. 151-155, 1991
- [8] L. P. Kaelbling, Hierarchical learning in stochastic domains: Preliminary results, *Proceedings of the International Conference on Machine Learning*, 1993
- [9] P. Dayan and G. E. Hinton, Feudal reinforcement learning, In *Advances in neural information processing systems 5* (C. L. Giles, S. J. Hanson and J. D. Cowan, ed), pp. Morgan Kaufmann, 1993
- [10] T. G. Dietterich, The maxq method for hierarchical reinforcement learning, *Proceedings of the 15th International Conference on Machine Learning*, 1998
- [11] R. S. Sutton, Learning to predict by the methods of temporal differences, *Machine Learning*, Vol. 3, pp. 9-44, 1988
- [12] G. A. Rummery and M. Niranjan, On-line q-learning using connectionist systems, Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994
- [13] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, MIT Press, 1998
- [14] A. H. Klopff, Brain function and adaptive systems - a heterostatic study, Technical Report AFCRL-72-0164, Air Force Cambridge Research Laboratories, Bedford, MA., 1972
- [15] S. Koenig and R. Simmons, G., Complexity analysis of real-time reinforcement learning, *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 1993
- [16] M. Wierling and J. Schmidhuber, Fast online $q(\lambda)$, *Machine Learning*, Vol. 33, pp. 105-115, 1998
- [17] S. Hirtle, C. and J. Jonides, Evidence of hierarchies in cognitive maps, *Memory and Cognition*, Vol. 13, pp. 208-217, 1985
- [18] R. Parr and S. Russell, Reinforcement learning with hierarchies of machines, In *Advances in neural information processing systems* (M. I. Jordan, M. J. Kearns and S. A. Solla, ed), pp. The MIT Press, 1997
- [19] B. L. Digney, Emergent hierarchical control structures: Learning reactive / hierarchical relationships in reinforcement environments, *Proceedings of the Fourth International Conference of Simulation of Adaptive Behavior*, 1996