JMLR: Workshop and Conference Proceedings 29:213–228, 2013 ACML 2013

# Q-learning for history-based reinforcement learning

**Mayank Daswani** mayank.daswani@anu.edu.au
**Peter Sunehag** Peter.Sunehag@anu.edu.au
**Marcus Hutter** Marcus.Hutter@anu.edu.au
*Research School of Computer Science,*
*Australian National University,*
*Canberra, ACT, 0200, Australia.*

## Abstract

We extend the Q-learning algorithm from the Markov Decision Process setting to problems where observations are non-Markov and do not reveal the full state of the world i.e. to POMDPs. We do this in a natural manner by adding $\ell_0$ regularisation to the pathwise squared Q-learning objective function and then optimise this over both a choice of map from history to states and the resulting MDP parameters. The optimisation procedure involves a stochastic search over the map class nested with classical Q-learning of the parameters. This algorithm fits perfectly into the feature reinforcement learning framework, which chooses maps based on a cost criteria. The cost criterion used so far for feature reinforcement learning has been model-based and aimed at predicting future states and rewards. Instead we directly predict the return, which is what is needed for choosing optimal actions. Our Q-learning criteria also lends itself immediately to a function approximation setting where features are chosen based on the history. This algorithm is somewhat similar to the recent line of work on lasso temporal difference learning which aims at finding a small feature set with which one can perform policy evaluation. The distinction is that we aim directly for learning the Q-function of the optimal policy and we use $\ell_0$ instead of $\ell_1$ regularisation. We perform an experimental evaluation on classical benchmark domains and find improvement in convergence speed as well as in economy of the state representation. We also compare against MC-AIXI on the large Pocman domain and achieve competitive performance in average reward. We use less than half the CPU time and 36 times less memory. Overall, our algorithm hQL provides a better combination of computational, memory and data efficiency than existing algorithms in this setting.

## 1. Introduction

Reinforcement Learning (RL) agents (Sutton and Barto, 1998) learn how to act well in an unknown environment through trial and error. In the case where the state is fully observed at every time step and the state space is finite, there are many algorithms that are guaranteed to eventually find the optimal policy. If the state space is small this also works well in practice and a near optimal policy can be found in reasonable time. Using a suitable exploration policy this time is polynomial in the relevant quantities (Strehl et al., 2009). One of the best known examples of an efficient MDP solving algorithm is Q-learning, which has the favorable properties of being efficient in both computation and memory as well as being able to learn about the optimal policy while following any sufficiently explorative policy. In this article we will extend Q-learning to a more realistic setting.

Unfortunately, in most real-world problems the states are only partially observed and the underlying state space may be large or even continuous. The feature reinforcement learning framework (Hutter, 2009; Nguyen et al., 2011, 2012; Daswani et al., 2012) is an example of history-based reinforcement learning (Mccallum, 1995) which attempts to remedy this by choosing a map from histories to states based on a cost function. The cost function needs to trade off between predictive ability and size of the resulting state space. An obvious idea is a complexity-penalised likelihood of the data experienced so far, but in complex domains with large observations spaces this may lead to an unnecessarily large state space and prohibitively slow learning. Therefore, the cost functions suggested by Hutter (2009) are not based on the likelihood of the observation sequence but instead of the state sequence resulting from the application of the map under consideration. However, if the map is injective, which is the case when one is using non-empty suffix trees, this is the same thing.

The above cost functions are inherently model-based and so unsuitable for function approximation. Our motivation is primarily to introduce a cost function that is value-based and model-free, hence scalable to a function approximation setting. We use the squared Q-learning error with a regulariser as our objective function in this feature RL setting. This value-based cost more discriminative and easily adapted to the function approximation setting. The resulting algorithm is an extension of Q-learning to the history-based setting. The objective function is simply complemented by an $\ell_0$ regularisation term and we optimise over not only the Q-values but also the choice of feature map. In the classical setting we have a fixed map and this regulariser therefore has no effect and the algorithm reduces to classical Q-learning. This objective is also similar to what is minimised by the Least Squared Fitted Q-iteration algorithm (Ernst et al., 2005; Farahmand et al., 2008).

The problem of reinforcement learning with function approximation in Markov Decision Processes where the full feature space is large compared to the number of samples, has recently been intensively studied. This problem has been addressed by the introduction of methods like regularised Least Squares Temporal Difference learning in (Ghavamzadeh et al., 2011; Kolter and Ng, 2009; Johns et al., 2010) where an $\ell_1$ regulariser promotes sparsity. If we do not have many more samples than features, then regularisation is a necessity. The Dantzig selector temporal difference method by Geist et al. (2012) uses an $\ell_\infty$ norm instead of $\ell_2$ for the error and an $\ell_1$ norm for parameter complexity.

Our primary difference from the classical function approximation setting is that we start with a small number of features and grow them over time via a simulated annealing procedure, rather than starting with a large set of all possible features and finding a sparse representation. In our history-based setting, if we want to be able to assume that the representation using all features is Markov like Ghavamzadeh et al. (2011) does, then we need a huge feature class that can capture all information in the history and this class grows over time. The size of the class of features forces us to use a stochastic search method over which features should be chosen as the non-sparse ones instead of an optimisation (like a gradient method) over the full feature vector. Such a stochastic search procedures works at least as easily with an $\ell_0$ norm which is directly the number of selected features as with the $\ell_1$ norm which can be understood as a substitute for $\ell_0$.

Our experiments on small domains show equal or better performance than CTΦMDP (Nguyen et al., 2011) both in terms of speed of convergence and economy of representation in the function approximation case. When compared against MC-AIXI (Veness et al., 2011)

on the large domain Pocman, the performance is comparable but with a significant memory and speed advantage.

After introducing background and notation in Section 2, we present our history based Q-learning algorithm in Section 3. In Section 4, we present our empirical evaluation and we conclude in Section 5 before describing our future plans in Section 6.

### 1.1. Related Work

The most closely related work is the feature reinforcement learning line (Hutter, 2009; Nguyen et al., 2011, 2012; Daswani et al., 2012). It defines agents with the same structure as our Q-learning agent but with a model-based cost function. The maps used in practice (such as suffix trees) are injective and therefore this cost simply becomes a penalised likelihood of the observations since the observation sequence can be recovered from the state sequence. Our model-free criteria is much more discriminative since it is only concerned with predicting expected return under an optimal policy. Furthermore, we extend the agent to use function approximation. The resulting agent is similar in form to the lasso-TD algorithms of (Ghavamzadeh et al., 2011; Kolter and Ng, 2009; Johns et al., 2010) but we use the optimal policy TD error instead of a policy evaluation setting and we use an $\ell_0$ regulariser instead of $\ell_1$. Another similar work for deterministic POMDPs is by Timmer and Riedmiller (2007) who use history lists to make an MDP state from the agent's history.

The Internal Policy State Gradient method by Aberdeen and Jonathan (2002) also uses a map from observations to histories (in their case finite state controllers (FSC)). In ISPG, the FSC is used to directly parameterise the policy space and then gradient ascent algorithms are used to find the best policy. In our case, we use the learned state space to find the optimal value function which then gives us a policy. Compared to ISPG, we have the advantage of not needing to fix the number of internal belief states but instead learn a suitable size.

## 2. Background

**Agent-Environment Framework.** The notation and framework is taken from (Hutter, 2004). An agent acts in an Environment $Env$ by choosing from actions $a \in \mathcal{A}$. It receives observations $o \in \mathcal{O}$ and real-valued rewards $r \in \mathcal{R}$ where $\mathcal{A}$,$\mathcal{O}$ and $\mathcal{R}$ are all finite. This observation-reward-action sequence happens in cycles indexed by $t = 1,2,3,\ldots$. We use $x_{1:n}$ throughout to represent the sequence $x_1 \ldots x_n$. The space of histories is $\mathcal{H} := (\mathcal{O} \times \mathcal{R} \times \mathcal{A})^* \times \mathcal{O} \times \mathcal{R}$. The history at time $t$ is given by $h_t = o_1 r_1 a_1 \ldots o_{t-1} r_{t-1} a_{t-1} o_t r_t$. The agent is then formally a (stochastic) function $Agent: \mathcal{H} \rightsquigarrow \mathcal{A}$ where $Agent(h_t) := a_t$. Similarly, the environment can be viewed as a (stochastic) function of the history, $Env: \mathcal{H} \times \mathcal{A} \rightsquigarrow \mathcal{O} \times \mathcal{R}$, where $Env(h_{t-1}, a_{t-1}) := o_t r_t$. A policy is then a map $\pi: \mathcal{H} \rightsquigarrow A$.

**Markov Decision Process (MDP).** If $Pr(o_t r_t | h_t, a_t) = Pr(o_t r_t | o_{t-1} a_t)$, the environment is said to be a discrete MDP (Puterman, 1994). In this case, the observations form the state space of the MDP. Formally an MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, R \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions and $R: \mathcal{S} \times \mathcal{A} \rightsquigarrow \mathcal{R}$ is the (possibly stochastic) reward function which gives the (real-valued) reward gained by the agent after taking action $a$ in state $s$. $\mathcal{T}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is the *state-transition function*. The agent's goal is to maximise its

expected future discounted reward, where a geometric discount function with rate $\gamma$ is used. In an episodic setting, the discounted sum is truncated at the end of an episode. The value of a state-action pair according to a stationary policy is given by $Q^\pi(s,a)=E_\pi\{R_t|s_t=s,a_t=a\}$ where $R_t=\sum_{k=0}^{t_{end}}\gamma^k r_{t+k+1}$ is the *return* and $t_{end}$ indicates the end of the the episode containing time $t$. In the non-episodic setting $R_t=\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}$. We want to find the optimal action-value function $Q^*$ such that $Q^*(s,a)=\max_\pi Q^\pi(s,a)$, since then the greedy policy with respect to $Q^*$ is optimal.

**ΦMDP.** Feature RL by Hutter (2009) is a framework that extracts features from the history that are useful in predicting future consequences of actions. It finds a map $\phi:\mathcal{H}\to\mathcal{S}$ such that the state at any time step $s_t=\phi(h_t)$ is approximately a sufficient statistic of the history. It uses a global cost function that is inspired by the minimum description length principle (Rissanen, 1978). The cost is the sum of the code lengths of state and reward sequences given actions. This cost is used within a global stochastic search technique (for example simulated annealing (Liu, 2008)) to find the optimal map. The standard cost is defined by

$$\text{Cost}(\phi|h_n):=CL(s_{1:n}|a_{1:n})+CL(r_{1:n}|s_{1:n},a_{1:n})+CL(\phi)$$

where $CL(s_{1:n}|a_{1:n})$ is the code length of the state sequence given the action sequence. The subsequence of states reached from a given state $s$ via action $a$ is i.i.d as it is sampled from an MDP. We form a frequency estimate of the model of this MDP. The code length is then the length of the arithmetic code with respect to the model plus a penalty for coding parameters. The coding is optimal by construction. $CL(r_{1:n}|s_{1:n},a_{1:n})$ follows similarly. $CL(\phi)$ is the code length of the description of the model itself, for example for a suffix tree it can be the length of an encoding of the tree, like in (Veness et al., 2011). However, since this is normally a small constant, it can safely be ignored in practical implementations. The Cost is well-motivated since it balances between coding states and coding rewards. A state space that is too large results in poor learning and a long state coding, while a state space that is too small can obscure structure in the reward sequence resulting in a long code for the rewards.

In this paper, CTΦMDP (Nguyen et al., 2011) refers to using the above ΦMDP algorithm along with simulated annealing over the space of context trees to find the best map, and then using approximate value iteration (AVI) to find the best policy. CTΦMDP can also use Q-learning to update the Q-values during the agent's interaction with the environment. Q-learning is also used in CTMRL by Nguyen et al. (2012) as a substitute for value iteration in large environments. However, both these methods are inherently model-based. Our contribution therefore is not being the first algorithm that can use Q-learning in this setting, but rather being completely model-free, with a cost function that is evaluated using Q-learning itself.

**CTMRL.** The Context Tree Maximising (CTM) for Reinforcement Learning (RL) algorithm (Nguyen et al., 2012) uses the CTM approach to sequence prediction that analytically finds the context-tree model by the minimum description length principle (Rissanen, 1978). The sequence prediction setting is adapted for RL by predicting the state-reward sequence conditioned on the actions. For large domains such as Pocman, the CTMRL approach binarises the percept (observation, reward) space, and additionally adds an "unseen" context, whose action value is initialised based on the value of the first subsequent seen state. Due

to the high space requirements of the CTM method in such environments the algorithm discards all the context tree maximisers (CTMs) at the start of every learning loop. New CTMs must then be created from only the history gained in the previous loop. This results in a significant loss in data efficiency.

**MC-AIXI CTW.** The Monte Carlo (MC) AIXI Context Tree Weighting (CTW) algorithm by Veness et al. (2011) is an approximation of the theoretically optimal universal agent AIXI (Hutter, 2004). Instead of using the universal mixture, it uses a mixture over all suffix trees with the weights being their code lengths. It dynamically creates the contexts so that only relevant ones are used. It uses the Krichevsky-Trofimov estimator to estimate the probabilities of symbols occurring in each context of the tree and by using properties of CTW (Wilems et al., 1995) it can calculate the probability of a history sequence in a computationally efficient way. So the (action-conditional) CTW tree maintains a model of the world, and then a Monte-Carlo Tree Search algorithm UCT by (Kocsis and Szepesvári, 2006) is used to determine which action to take next, by using the current mixture of suffix trees as a generative model.

## 3. Q-learning for history-based RL

In this section, we extend the Q-learning algorithm(Watkins, 1989) to non-Markov environments.

Q-learning aims to find the fixed point of the Bellman equation $Q=TQ$ where $T$ is the Bellman operator in an online fashion using the update rule

$$Q(s,a) \leftarrow Q(s,a)+\alpha_t \Delta_t$$

where $\Delta_t$ is the temporal difference

$$\Delta_t = r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t)$$

Q-learning is an off-policy algorithm that asymptotically converges to the optimal action-value function $Q^*$ given sufficient exploration and a learning rate ($\alpha$) that satisfies the Robbins-Monro(Robbins and S.Monro, 1951) conditions. It is simple to implement and works well in practice. However, Q-learning in the above form cannot be used in a history-based RL setting, since we do not have the state space.

For each $\phi$ we define a Q-table based on the state space given by $\phi$. We denote this Q-table by $Q^\phi : \mathcal{H} \times \mathcal{A} \rightarrow \mathbb{R}$ and it is of the form $Q(\phi(h),a)$. We use the squared pathwise Q-learning error to find a suitable map $\phi : \mathcal{H} \rightarrow \mathcal{S}$ by selecting $\phi$ to minimise the following cost,

$$\text{Cost}_{QL}(\phi) = \min_{Q_\phi} \frac{1}{2} \sum_{t=1}^{n} (\Delta_t^{Q_\phi})^2 + Reg(\phi)$$

This is similar to the objective function in Regularised Least Squares Fitted Q-iteration (Farahmand et al., 2008). However they have a penalty (needed due to their non-parametric setting) based on the smoothness of the (continuous) Q-function, whereas our regulariser is on $\phi$ which is fixed in their setting. In particular, we use the regulariser $Reg(\phi) =$

$\frac{\beta}{2}|\mathcal{S}||\mathcal{A}|\log_2(n)$ where $\beta\in\mathbf{R}$ is a regularisation constant and $n$ is the length of the history so far.

This cost also easily extends to the linear function approximation case where we approximate $Q(\phi(h_t),a_t)$ by $\xi(h_t,a_t)^T w$ where $\xi:\mathcal{H}\times\mathcal{A}\to\mathbb{R}^k$ for some $k\in\mathbb{R}$.

$$\mathrm{Cost}_{QL}(\xi)=\min_w \frac{1}{2}\sum_{t=1}^{n}\Big(r_{t+1}+\gamma\max_a \xi(h_{t+1},a)^T w-\xi(h_t,a_t)^T w\Big)^2 + Reg(\xi)$$

The regulariser on $\xi$ is analogously $Reg(\xi)=\frac{\beta}{2}k\log_2(n)$.

Let $\frac{1}{n}\sum_{t=1}^{n}\Delta_t^2:=\delta_n$. Let us also consider the tabular case, where $\phi:\mathcal{H}\to\mathcal{S}$ and assume the state space produced is an MDP. Fix a policy $\pi$. Then $Pr(s_t=s,a_t=a)=Pr(s_t=s,a_t=\pi(s_t))$. Let $d_{s,a}=\lim_{n\to\infty}\frac{1}{n}\sum_{t=1}^{\infty}Pr(s_t=s,a_t=a)$. For a fixed policy, $d_{s,a}$ always exists and is the expected proportion of time you spend in each state and select a particular action. We can express $Cost_{QL}$ in the following way, where the expectations are taken over the next state.

$$\begin{aligned}
&\lim_{n\to\infty}E[\delta_n]=\lim_{n\to\infty}\frac{1}{n}\sum_{t=1}^{n}E[\Delta_t^2]\\
&=\lim_{n\to\infty}\frac{1}{n}\sum_{t=1}^{n}\sum_{s,a}Pr(s_t=s,a_t=a)E[\Delta_t^2|s_t=s,a_t=a]\\
&=\sum_{s,a}E[R(s,a,s')+\gamma\max_{a^*}Q(s',a^*)-Q(s,a)|s,a]^2(\lim_{n\to\infty}\frac{1}{n}\sum_{t=1}^{n}Pr(s_t=s,a_t=a))\\
&=\sum_{s,a}d_{s,a}\Big(E[R(s,a,s')+\gamma\max_{a^*}Q(s',a^*)-Q(s,a)|s,a]^2\\
&+Var(R(s,a,s')+\gamma\max_{a^*}Q(s',a^*)|s,a)\Big).
\end{aligned}$$

The expression $\sum_{s,a}d_{s,a}E[R(s,a,s')+\gamma\max_{a^*}Q(s',a^*)-Q(s,a)|s,a]^2$ is the mean squared Bellman Error (MSBE) expressed for Q-values. In our setting we also vary the feature map $\phi$ which in turn affects the policies that can be represented. i.e. we pick the $\phi$ that minimises the MSBE along with the non-vanishing variance term. A smaller variance means we have better predictive information in our state representation, and the growth of the representation is tempered by our regulariser.

In practice for a fixed $\phi$ we use Q-learning to find the weights that satisfy the fixed point equation which in the tabular case is (trivially) identical to minimising the Bellman error. In the function approximation case, Q-learning in it's most naive form is not guaranteed to converge and the fixed point and Bellman error solutions are different. The recently proposed algorithm Greedy-GQ by Maei et al. (2010) instead minimises the projected Bellman error via approximate gradient descent. The convergence guarantees only apply to the case where the behaviour policy is fixed, which is unfortunately not true in our setting, but the algorithm could still be used. For this paper however, we use standard Q-learning and on the examples we run it does not diverge.

The cost we suggest here can, just as the original suggestions by Hutter (2009), be interpreted using code lengths. One term is coding the reward sequence $r_t$ by a Gaussian

distribution with mean $Q(s_t,a_t)-\max_{a'}Q(s_{t+1},a')$. The regulariser codes the $Q(s,a)$ values and in the tabular setting is proportional to the number of parameters to learn for the map under consideration. Therefore, it is a regularisation constant times an $\ell_0$ norm. The regularisation constant should be larger when we have more samples since the parameter estimates are more accurate then and deserve to be coded with higher accuracy as is argued by Hutter (2009) where the dependence, like here, is logarithmic in the length of the history.

**Cost consistency.** A theory similar to the one by Sunehag and Hutter (2010) for the model-based Cost function with similar assumptions and conclusions can be developed for hQL. If a map $\phi$ together with the taken actions generate a sequence such that the frequency of each $(s,a,s',r)$ converges asymptotically (conditions for this related to ergodicity and the form of the maps are studied by Sunehag and Hutter (2010)) then (trivially) $\min_Q \frac{1}{n}\sum_{t=1}^{n}(\Delta_t^Q)^2$ converges to an expectation $\Gamma_\phi = \min_Q E_{s,a,s',r}(r+\gamma\max_a' Q(s',a')-Q(s,a))^2$. If the class is finite and only consists of such $\phi$ and if we use a sublinearly growing regulariser term (here logarithmic), i.e. $\frac{Reg}{n}\to 0$, then we eventually only choose $\phi^* \in \operatorname{argmin}_\phi \Gamma_\phi$. This means that we choose a map for which minimal Q-learning error can be achieved. This reasoning (as in (Sunehag and Hutter, 2010)) applies to any cost function which are sums of terms depending on $(s,a,s',r)$ quadruples. However, in our algorithm the policy changes in such a way that we cannot guarantee these conditions. Also, good finite time behaviour demands the regulariser to relate to the estimate error in the other term in a suitable manner as in the line of work by Ghavamzadeh et al. (2011); Kolter and Ng (2009); Johns et al. (2010) performing finite-time analysis for regularised LSTD. Analysing this for hQL is future work that will be more meaningful than the stated asymptotic result.

**Agent.** Algorithm 1 defines a ΦMDP agent. The agent is given some number of initial random actions and a starting map. It then acts according to the fixed (optimal) policy found by Q-learning according to the current map for $M$ iterations (an epoch). At the start of each epoch, the agent has the opportunity to change it's map via a simulated annealing process. The $\text{Cost}_{QL}(\phi)$ is used as the cost in this simulated annealing procedure in Algorithm 2. Algorithm 2 also requires a neighbourhood function which we define separately for each feature class. We could use any off-policy method to find the policy in Algorithm 1, but using Q-learning itself is self-consistent. Note that we cannot use an on-policy method here as the map can change throughout the history, and the policy learnt depends on the map. Therefore at a particular time, the agent's history is one that is generated by a series of changing policies.

**Features.** In our experiments we use the feature class of suffix trees in the tabular case, and new feature class *event selector* in the function approximation case. Suffix trees are a popular choice in history-based learning since they can be computationally efficient and naturally capture short-term memory. They have been used in (Mccallum, 1995; Veness et al., 2011; Nguyen et al., 2011) and others.

**Definition 1 (Suffix Tree)** *(from (Daswani et al., 2012)) Let $\mathcal{O}=\{o^1,o^2,o^3,...,o^d\}$ be a d-ary alphabet. A suffix tree is a d-ary tree in which the outgoing edges from each internal node are labelled by the elements of $\mathcal{O}$. Every suffix tree has a corresponding suffix set which is the set of strings $\mathcal{S}=\{s^1,s^2,...,s^n\}$ generated by listing the labels on the path from each leaf node to the root of the tree.*

No string in the suffix set is a suffix of any other string and any sufficiently long string must have a suffix in the set. The *l-th* level of the tree corresponds to the *l-th* last observation in the history. Thus any history of sufficient length must be mapped to one and only one member of the suffix set, and each member is therefore called a state. A neighbour of a suffix tree is either a split of a leaf node in the tree to form $|\mathcal{O}|$ more children, or a merging of the $|\mathcal{O}|$ children leaf nodes to the single parent node.

An event selector is a set of features $\xi_j$. Each feature $\xi_j$ consists of a position $m$ and an observation $o$. Feature $\xi_j$ is on (i.e. equal to 1) if the $(n-m)$-th position in the history has observation $o$. More formally we use Definition 2.

**Definition 2 (Event selector)** *Let $o_i$ be the i-th observation in $h_{1:n}$. An event selector is a set $\xi=\{\xi_1,\xi_2,...,\xi_k\}$ where $\xi_j:\mathbb{N}\times\mathcal{O}\times\mathcal{O}^n\to\mathbb{B}$ is a function such that $\xi_j(m,o,o_{1:n})=1$ if $o_{n-m}=o$ and 0 otherwise.*

The neighbour of an event selector is either the addition or removal of a feature to the set. Note that we can similarly define an event selector for observation-action pairs, but we use the definition above for this paper.

The bit selector is a modification of the event selector. Instead of picking out whether the $(n-m)$-th position in the history has observation $o$ we check whether the $c$-th bit of $o_{n-m}$ is 0 or 1. This approximator is particularly useful in dealing with environments where the individual bit structure has relevance (such as Pocman). This binarisation is similar to those performed by MC-AIXI (Veness et al., 2011) and CTΦMDP (Hutter, 2009).

**Definition 3 (Bit selector)** *Let $bin(o_i)$ be the binarisation of the i-th observation in $h_{1:n}$. Let $bin(o_i)^c$ represent the c-th bit of this binary number. A bit selector is a set $\xi=\{\xi_1,\xi_2,...,\xi_k\}$ where $\xi_j:\mathbb{N}\times\mathbb{N}\times\mathbb{B}\times\mathcal{O}^n\to\mathbb{B}$ is a function such that $\xi_j(m,c,b,o_{1:n})=1$ if $bin(o_{n-m})^c=b$ and 0 otherwise.*

## 4. Experiments

In this section we describe the experimental setup and the domains used to evaluate the new cost. On the smaller domains we test three algorithms : history Q-learning (hQL) using suffix trees, the function approximation of hQL using event selector features (FAhQL) and the original ΦMDP using suffix trees. ΦMDP has proven to be competitive (Nguyen et al., 2011, 2012) against MC-AIXI (Veness et al., 2011), and better than both Active LZ (Farias et al., 2007) and U-tree (Mccallum, 1996).

Every experiment was run 30 times. The agent was given an initial history produced by taking random actions. Each run of an experiment was conducted over 100 epochs. Each epoch contains 100 iterations of the agent performing actions according to its current policy, based on the current map. Additionally the agent uses $\epsilon$-exploration until a certain number of epochs is completed. After the completion of an epoch, the agent is given a chance to change current map via the simulated annealing procedure. The annealing procedure uses an exponential cooling function. Plots show every 5th point with 2 standard error on either side. The exact constants used for all the experiments can be found in Table 2. When using Q-learning (both within the cost and without) we use several runs through the data such

**Algorithm 1:** A high-level view of the generic ΦMDP algorithm.

**Input** : Environment $Env()$
Initialise $\phi$
Initialise history with observations and rewards from $t{=}init\_history$ random actions
Initialise $M$ to be the number of timesteps per epoch
**while** *true* **do**
  $\phi{=}SimulAnneal(\phi,h_{1:t})$
  **for** $t$ = *1 to* $n$ **do**
    $s_t{=}\phi(h_t)$
  **end**
  $\pi{=}Qlearn(s_{1:t},r_{1:t},a_{1:t-1})$
  **for** $i{=}1,2,3,...M$ **do**
    $a_t \leftarrow \pi(s_t)$
    $o_{t+1},r_{t+1} \leftarrow Env(h_t,a_t)$
    $h_{t+1} \leftarrow h_t a_t o_{t+1} r_{t+1}$
    $t \leftarrow t{+}1$
  **end**
**end**

that it converges. We found that for the small domains we test on $\epsilon$-exploration was enough to ensure convergence to optimality.

We mildly tune the parameters by running the algorithm for a small number of iterations on each environment. The FAhQL and hQL constants are very similar. The choice of regularisation constant appears to have some dependence on the stochasticity of the environment.

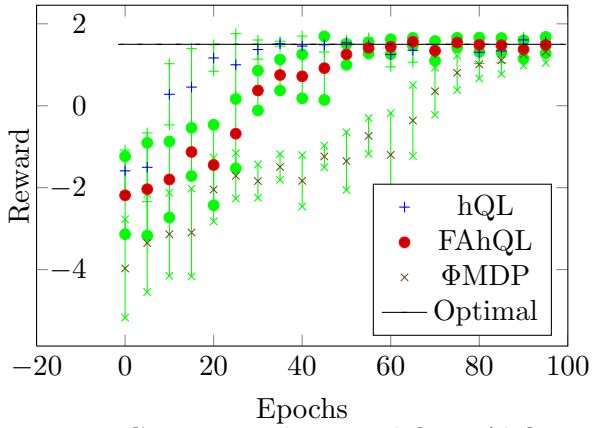


Figure 1: Comparison between hQL, FAhQL and ΦMDP on Cheese Maze

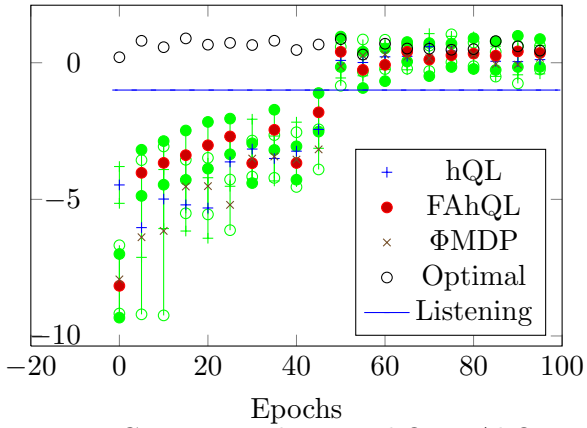


Figure 2: Comparison between hQL, FAhQL and ΦMDP on Tiger

**Tiger.** The Tiger domain is familiar in the partially observable reinforcement learning literature (Veness et al., 2011). There are two doors and behind one door is a tiger, while the other hides a pot of gold. The agent must decide which door to open. The agent has an

**Algorithm 2:** Simulated Annealing Search

```
SimulAnneal()
Input :
    getNeighbour() : A neighbour function providing the next map
    schedule() : The cooling scheme
    initialMap : The starting map in the search
    Cost() : A cost function
    h : The history sequence so far
currentMap = bestMap := initialMap
currentCost = bestCost := Cost(h,initialMap)
for t ← 1 to N do
    candidateMap ← currentMap.getNeighbour()
    candidateCost ← Cost(h, candidateMap)
    δ ← currentCost - candidateCost
    T ← schedule(t)
    p ←uniform(0,1)
    if δ> 0 or p <e^δ/T then
        currentMap ← candidateMap
        currentCost ← candidateCost
        if currentCost < bestCost then
            bestMap ← currentMap
            bestCost ← currentCost
        end
    end
end
```

**Algorithm 3:** Cost function

```
Cost()
Input:
    φ : The current map.
    h_n : The history sequence so far of size n.
for t = 1 to n do
    s_t = φ(h_t)
end
w = Qlearn(s_{1:n}, r_{1:n}, a_{1:n-1}) (can go through the history multiple times.)
Calculate currentCost = Cost_QL(w, φ_{1:n}, r_{1:n}, a_{1:n-1}).
Return currentCost
```

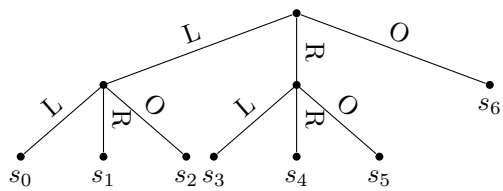


Figure 3: Best chosen suffix tree for tiger

| Position | Observation |
|---|---|
| 1 | L |
| 1 | R |
| 1 | O |
| 2 | L |
| 2 | R |

Figure 4: Best chosen event selector for tiger

The observations L,R and O refer to Left, Right and Open (Door) respectively. The suffix tree on the left shows that the agent remembers two observations in the past of listening in order to make it's decision about which door to open. With a linear function approximator however, this can be represented more compactly as seen in the table on the right.

action Listen available to it that says with 0.85 probability which door the tiger is behind. Choosing to Listen has a penalty of -1. The episode ends when the agent chooses a door. If it chooses the door with the pot of gold it receives a reward of 10, or else it is eaten by the tiger for a penalty of -100. We also generate the optimal policy for tiger which is to wait until the last two listens agree and plot that on the graph. The graph showing the optimal policy is not constant due to the fact that each epoch does not necessarily end at the end of an episode, hence some rewards can be carried to the next epoch.

From Figure 2 we see a very similar performance from the three algorithms. It should be noted that the suffix tree based algorithms hQL and ΦMDP found features of size 21(7 states, 3 actions) to be optimal, the function approximator generally uses between 15 and 18 features (although sometimes goes up to 24). The features used by the linear function approximator for optimal performance in the Tiger problem are shown in Figure 4, with the suffix tree (having 21 features) is shown in Figure 3.

**Cheese Maze.** The cheese maze domain (Veness et al., 2011) examines the issue of state aliasing. The agent starts in any of the available positions in the maze. The observation it receives is given by the number on the square (see Figure 5). The agent's task is to find the cheese. Each valid move it makes costs -1, hitting a wall is penalised -10, and eating the cheese is rewarded 10. Once the agent eats the cheese the episode restarts.

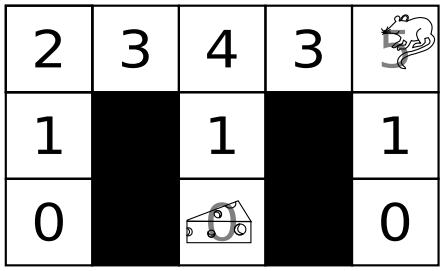


Figure 5: Cheese maze

On this domain there are clearer differences between the algorithms. hQL converges the fastest, while ΦMDP converges only in the last few epochs. We observe that hQL at best uses 64 features (16 states, 4 actions) while FAhQL generally converges to using 36 features.

In the above experiments we see some sudden drops in the reward (for e.g. hQL at epoch 20 in Figure 2). These dips are generally formed by the algorithm changing its map. A new map implies new states and the algorithm does not necessarily know how to act optimally in these new states given the data it currently has.

Table 1: Computational comparison on Pocman

| Agent | Cores | Memory(GB) | Time(hours) | Iterations |
|---|---|---|---|---|
| MC-AIXI 96 bits | 8 | 32 | 60 | $1 \cdot 10^5$ |
| MC-AIXI 48 bits | 8 | 14.5 | 49.5 | $3.5 \cdot 10^5$ |
| FAhQL | 1 | 0.4 | 17.5 | $3.5 \cdot 10^5$ |

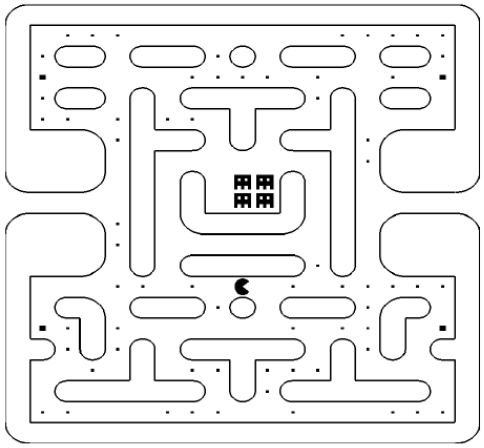

Figure 6: Pocman Domain

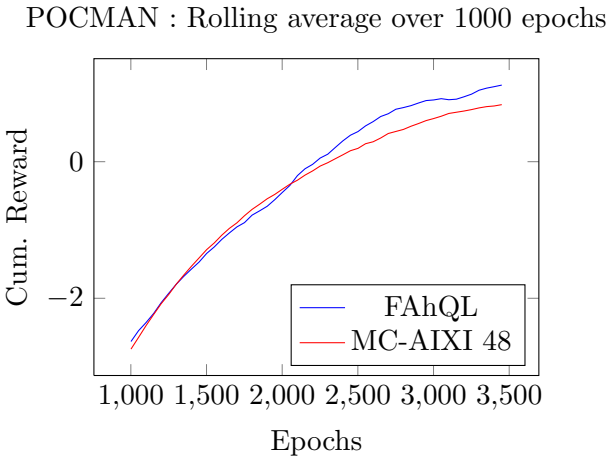


Figure 7: MC-AIXI vs hQL on Pocman

**Pocman.** The Partially Observable Pacman domain was introduced in (Veness et al., 2011). The agent starts in the center of a standard Pacman map (17x17), see Figure 6. At every timestep it receives a bit sequence containing the following bits. 4 bits to code whether there is a wall in an adjacent square, 4 bits to code whether there is food in an adjacent square, 4 bits to check if there is a ghost in any **direction**, 3 bits to "smell" food within 2, 3 and 4 squares and 1 bit that is active when the agent has swallowed a power up pill. It receives a -1 reward every time it makes a valid move. If it attempts to move into a wall it receives -10. Eating a food pellet gains 10 and eating all the food on the map gains 100. Eating a ghost resets the ghost to the center of the map. Being eaten by a ghost ends the episode and it receives a reward of -50. All rewards are cumulative, which means that if it moved and ate a pellet it receives a reward of 9 for that timestep. We treat the Pocman domain as a non-episodic discounted task.

For this domain, we change the experimental setup. We now test FahQL with a bit selector feature space. It is only feasible to do one run so the graphs show rolling average over the previous 1000 epochs (or 100000 iterations). We no longer run through the Q-learning multiple times, and while initial estimates might be inaccurate we use enough data that this is not a problem, and we change the map every 200 iterations rather than 100. After 100000 iterations we delete 10000 timesteps of data from the start of the history, so that our history is now always 100000 timesteps long (i.e. we consume a constant amount of memory). This helps both speed and memory efficiency, and the number of timesteps is enough to ensure convergence. We compared against MC-AIXI which is still the best performing algorithm on the domain. In order to make this comparison fair we use the

Table 2: Constants used for each experiment for hQL and FAhQL.

| Experiment | $\alpha$ | $\beta$ | $\epsilon$ | *amaxs* | *astartT* | *acoolRate* |
|---|---|---|---|---|---|---|
| hQL Tiger | 0.01 | 1.5 | 0.1 | 10 | 4,000 | $5{\cdot}10^{-2}$ |
| hQL Cheese | 0.01 | 0.02 | 0.1 | 10 | 5,000 | $5{\cdot}10^{-2}$ |
| FAhQL Tiger | 0.01 | 1 | 0.1 | 20 | 7,000 | $5{\cdot}10^{-2}$ |
| FAhQL Cheese | 0.01 | 0.02 | 0 | 10 | 5,000 | $5{\cdot}10^{-2}$ |
| FAhQL Pocman | 0.001 | 2 | decay | 20 | 4,000 | $5{\cdot}10^{-7}$ |

*Common to all experiments was* $\gamma$=0.99, init-history *the number of initial random actions performed by the agent at 200 and* stop-explore=*50 is the epoch beyond which the agent no longer uses* $\epsilon$*-exploration (N/A for Pocman).* $\alpha$ *is the learning rate,* astartT *refers to the starting temperature T in the cooling schedule,*acoolRate *to the decay of the exponential cooling schedule and* amaxs *refers to the number of timesteps allowed in the annealing procedure.*

same setup as in Veness et al. (2011) with an exploration rate starting at 0.9999 that geometrically decreases at a rate of 0.99999 per timestep. CTΦMDP cannot deal with such large observation spaces due to the large explosion in number of trees it must consider. The context-tree maximising algorithm upon which CTMRL is based is a variant of CTW and consumes a large amount of memory. CTMRL also needs many ad-hoc manipulations to run on Pocman. These included sacrificing data efficiency to avoid memory problems by discarding the CTMs in each learning loop, making the algorithm space and relatively time efficient but needing 100 million iterations.

Unfortunately we could not run MC-AIXI with the exact parameter settings as in Veness et al. (2011) due to time and memory constraints (see Table 1). The furthest that the 96 bit, 4 look ahead algorithm went was to a 100000 iterations before running out of memory. We instead ran MC-AIXI using 48-bit (which is 2 percepts in the past) and 2 look-ahead. On the PACMAN domain this shorter memory and horizon can even be an advantage, given that in order to get a good behaviour one only needs to avoid walls, ghosts and eat food in adjacent squares which only needs one or two observations in the past, and additional observations complicate the problem.

FAhQL used 400MB of memory and finished 350000 iterations in about 17 hours. It was allowed to look up to 4 observations in the past (96 bits in MC-AIXI terms). It shows comparable performance to MC-AIXI (48-bits) with over 36 times the memory efficiency and about 3 times the speed despite being single-threaded. It also has a much more significant speed-up over the 96-bit MC-AIXI but this is harder to quantify given our data. We should note that we can regulate the speed by controlling the epoch length.

## 5. Conclusion

We extended the Q-learning algorithm from MDPs to a general reinforcement learning setting. The resulting algorithm can be understood as a feature reinforcement learning algorithm, but with a more discriminative model-free cost function. The original CTΦMDP algorithm (Nguyen et al., 2011) is incapable of dealing with large observation spaces effec-

tively whilst the CTMRL agent needed many ad-hoc modifications to work on the large Pocman domain and due to that suffered bad data efficiency. This has been a motivation for this work on finding an algorithm that works naturally with function approximation and which is as discriminative as possible. Our algorithm lends itself naturally to a function approximation setting. Our empirical evaluation shows some improvement in convergence speed on classical POMDP benchmark domains with function approximation resulting in more economical feature vector sizes. We demonstrated our performance on a large domain Pocman where we performed competitively against MC-AIXI while using 20 times less memory. Our algorithm hQL provides computational efficiency at least on par with CTMRL (given the same number of map re-estimation points) while retaining the data efficiency of MC-AIXI and superior memory efficiency to both.

## 6. Future Work

There are several optimisations that can improve the computational efficiency of the algorithm that we will investigate. We note that hQL also naturally works with both continuous environments as well as continuous features. The ability to deal with continuous history-based problems in a linear architecture also allows us to attempt to learn the belief states of a POMDP without learning the underlying MDP structure. Furthermore, we are also interested in performing a finite time analysis.

**Acknowledgements.** The research was partly supported by the Australian Research Council Discovery Project DP120100950.

## References

D. Aberdeen and B. Jonathan. Scaling internal-state policy-gradient methods for pomdps. In *Proc. ICML-02, pp.310*, pages 3–10, 2002.

M. Daswani, P. Sunehag, and M. Hutter. Feature reinforcement learning using looping suffix trees. *JMLR Workshop and Conference Proceedings : EWRL 2012*, 24:11–24, 2012.

D. Ernst, P. Geurts, L. Wehenkel, and L. Littman. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

A. Farahmand, M. Ghavamzadeh, C. Szepesvri, and S. Mannor. Regularized fitted q-iteration: Application to planning. In Sertan Girgin, Manuel Loth, Rmi Munos, Philippe Preux, and Daniil Ryabko, editors, *Recent Advances in Reinforcement Learning*, volume 5323 of *Lecture Notes in Computer Science*, pages 55–68. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-89721-7.

V. F. Farias, C. C. Moallemi, T. Weissman, and B. Van Roy. Universal reinforcement learning. *IEEE Transactions on Information Theory*, 56(5):2441–2454, 2007.

M. Geist, B. Scherrer, A. Lazaric, and M. Ghavamzadeh. A Dantzig selector approach to temporal difference learning. In *In International Conference on Machine Learning*, 2012.

M. Ghavamzadeh, A. Lazaric, R. Munos, and M. W. Hoffman. Finite-sample analysis of lasso-td. In Lise Getoor and Tobias Scheffer, editors, *ICML*, pages 1177–1184. Omnipress, 2011.

M. Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability.* Springer, Berlin, 2004.

M. Hutter. Feature reinforcement learning: Part I: Unstructured MDPs. *Journal of Artificial General Intelligence*, 1:3–24, 2009. ISSN 1946-0163.

J. Johns, C. Painter-wakefield, and R. Parr. Linear complementarity for regularized policy evaluation and improvement. In *In Advances in Neural Information Processing Systems 23*, 2010.

L. Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *In* The $17^{th}$ European Conference on Machine Learning, pages 99–134, 2006.

J. Z. Kolter and A. Y. Ng. Regularization and feature selection in least-squares temporal difference learning. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 521–528, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553442.

J. S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer, corrected edition, January 2008. ISBN 0387952306.

H.R. Maei, C. Szepesvári, S. Bhatnagar, and R. S. Sutton. Toward off-policy learning control with function approximation. In Johannes Fürnkranz and Thorsten Joachims, editors, *ICML*, pages 719–726. Omnipress, 2010. ISBN 978-1-60558-907-7.

A. K. Mccallum. Learning to use selective attention and short-term memory in sequential tasks. In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pages 315–324. MIT Press, 1996.

R. Andrew Mccallum. Instance-based utile distinctions for reinforcement learning with hidden state. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 387–395. Morgan Kaufmann, 1995.

P. M. Nguyen, P. Sunehag, and M. Hutter. Context tree maximizing. In Jörg Hoffmann and Bart Selman, editors, *AAAI*. AAAI Press, 2012.

P.M. Nguyen, P. Sunehag, and M. Hutter. Feature reinforcement learning in practice. *Proceedings of European Workshop on Reinforcement Learning, Athens, Greece*, 2011.

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994. ISBN 0471619779.

J. J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

H. Robbins and S.Monro. A stochastic approximation method. *Annals of Mathematical Statistics 22*, pages 400–407, 1951.

A. L. Strehl, Lihong Li, and M. L. Littman. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research.*, 10:2413–2444, dec 2009. ISSN 1532-4435.

P. Sunehag and M. Hutter. Consistency of feature Markov processes. In *Proc. 21st International Conf. on Algorithmic Learning Theory (ALT'10)*, volume 6331 of *LNAI*, pages 360–374, Canberra, 2010. Springer, Berlin. ISBN 978-3-642-16107-0. doi: 10.1007/978-3-642-16108-7_29.

R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

S. Timmer and M. Riedmiller. Safe q-learning on complete history spaces. In *Proceedings of the 18th European Conference on Machine Learning*, ECML '07, pages 394–405, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74957-8. doi: 10.1007/978-3-540-74958-5_37.

J. Veness, K. S. Ng, M. Hutter, W. Uther, and D. Silver. A Monte Carlo AIXI approximation. *Journal of Artificial Intelligence Research*, 40:95–142, 2011. ISSN 1076-9757. doi: 10.1613/jair.3125.

C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989.

F.M.J. Wilems, Y.M. Shtarkov, and T.J. Tjalkens. The context tree weighting method: Basic properties. In *IEEE Transactions on Information Theory*, 1995.