Safe State Abstraction and Discounting in Hierarchical Reinforcement Learning

Bernhard Hengst
National ICT Australia
Computer Science and Engineering
University of New South Wales, Sydney 2052 Australia
bernhardh@cse.unsw.edu.au

UNSW CSE TR 0308

Abstract

The great benefit in state abstraction for hierarchical reinforcement learning (HRL) is the potential improvement in computational complexity with significant compaction of the value function. Safe state aggregation of reusable sub-task states is not possible in general for a decomposed MDP using one decomposed discounted cumulative reward function. This severely limits the effectiveness of HRL, particularly for infinite horizon problems. This paper makes two related and novel contributions: (1) the introduction of an additional supporting decomposed discount function allowing state abstraction in the face of discounting and (2) modifications to adapt HRL to solve infinite horizon problems in which the recursively optimal policy may require a sub-task to persist.

1 Introduction

State abstraction has been shown to be critical for scaling reinforcement learning [1, 2, 3, 4]. In many cases, without state abstraction, hierarchical reinforcement learners take longer to converge than flat learners, but converge much faster than flat learners if state abstraction can be introduced.

Multi-time models for options [5] and similar approaches for MAXQ [3], HAMs [6] and ALisp [4] are used to apply the correct discount on the termination of abstract actions. From these models we know that the value of a base level state in a sub-task is jointly dependent on the value of the state reached after terminating the sub-task and the time to termination.

Safely state abstracting reusable sub-tasks is severely limited for discounted cumulative reward problems. These hierarchical reinforcement learners decompose the value function into the value accumulated inside the sub-task and the value after termination. By safe state abstraction we mean that the value function over all states for any policy is the same before and after abstraction. To safely abstract reusable sub-task states we require for any stationary policy that (1) the value after termination from all sub-task states is the same in a particular context in which the sub-task is invoked, and (2) the state value inside the sub-task is not effected by the value after termination in any context in which the sub-task is invoked. These conditions cannot be met in general using a single decomposed value function because discounting makes the components of the value function inside and outside the sub-task interdependent. Dietterich also discusses this issue under *result distribution irrelevance* [3] concluding that this condition is applicable only in an undiscounted setting.

The consequences, therefore, of discounting in HRL present a major impediment to scaling. In particular, state abstraction is prevented for infinite horizon problems where discounting is mandatory for cumulative reward value functions.

The first contribution of this paper is to show how the introduction of an additional decomposed *discount function* working in concert with a decomposed value function allows safe state abstraction in the face of discounting. This approach is a natural extension to the automatic decompositions produced by HEXQ [7] and can be applied to other HRL methods with additional constraints.

The second and related contribution is to show how some hierarchically decomposed infinite horizon MDPs can now be solved using safe state abstraction. MAXQ sub-tasks, Options and machines for HAMs assume termination of the related abstract action. By allowing non-terminating abstract actions and introducing an additional pseudo reward function, a recursively optimal solution can be found to some infinite horizon MDPs even if the solution requires continuing in a sub-task.

Section 2 derives the decomposition equations for a HEXQ decomposed MDP and shows how state abstraction can be retained with discounting. Section 3 explains how HRL can be modified to solve infinite horizon problems. Section 4 demonstrates HRL on an infinite horizon taxi task requiring abstract actions to persist and on a larger infinite horizon soccer problem that is intractable on present day desktop computers without state abstraction.

2 State Abstraction and Discounting

We will first review the definition of a HEXQ partition for a factored MDP [7] and show that for every HEXQ sub-MDP the number of steps to termination is independent of the expected value at termination. We will then derive the decomposition equations to safely state abstract a discounted value function.

The starting point is a standard MDP (S, A, T, R) where S is a finite set of states and A a finite set of actions. When transitioning from state s to s' on action a the transition and immediate reward functions are $T^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, _t = a\}$ and $R^a_{ss'} = E\{r_{t+1} | s_{t+1} = s', s_t = s, a_t = a\}$ respectively. With the states $s \in S$ partitioned into regions $g \in G$ an *exit* from region g is a state-action pair (s^o, a) such that taking action a from state $s^o \in g$ may reach a state not in g (or the MDP may terminate) in a single transition. State s^o is referred to as the *exit state*. The state reached after exiting is called an *entry state* to the next region or a *termination state* from the current region, depending on our perspective. Assume that the states are factored by two variables such that $s = (x, y)$.

A *HEXQ Partition* G of the states $s = (x, y)$ of a MDP into regions g is defined by the conjunction of the following conditions for all $g, g' \in G$

1. Same Context : all states in a region must have the same y label:
for all $s = (x, y),\ s' = (x', y') \in g$
$y = y'$

2. Markov transitions : all states with the same x label in different regions must have similar reward and transition functions to states inside their regions:
for all $a \in A$, $s = (x, y),\ s' = (x', y) \in g$ and $t = (x, y'),\ t' = (x', y') \in g'$
$T^a_{ss'} = T^a_{tt'}$ and $R^a_{ss'} = R^a_{tt'}$

3. Reachable exits : all exit states must be reachable within the region from all entry states, with probability one, under some stationary policy.

The HEXQ partition ensures regions with the same set of x labels are isomorphic in the sense that they have equivalent internal Markov properties with respect to state transitions and rewards. HEXQ parameterises a set of sub-MDPs over the x label region classes so that each sub-MDP is allowed to reach only one possible exit state. We have implicitly introduced state abstraction across the region class by ignoring the y labels. This first type of state abstraction is similar to model minimisation [8] for each region. The decomposition of the value function in HEXQ was inspired by MAXQ. For HEXQ the local reward functions for sub-MDPs do not include the primitive reward received on exiting the sub-MDP. The HEXQ partitioning can be applied recursively resulting in a hierarchy of sub-MDPs not unlike a MAXQ graph. The value of a state s under a stationary policy π in a sub-MDP m is

$$V^\pi_m(s) \quad = \quad V^\pi_{m-1}(s) + \sum_{s',N} P^\pi(s', N | s, \pi(s)) \gamma^{N-1} [R + \gamma V^\pi_m(s')] \qquad (1)$$

Parameter γ is the discount factor, N is a random variable representing the number of steps required to exit the next lower level sub-MDP which we denote as $m - 1$. $V^\pi_{m-1}(s)$ is the internal value of state s in sub-MDP $m - 1$. $P^\pi_m(s', N | s, \pi(s))$ is the joint probability of reaching state s' after termination of sub-MDP $m - 1$ in N steps starting in state s. R is the expected primitive reward on exit.

Importantly, for a HEXQ partition, the termination state s' reached and the expected reward R on exit are independent of the number of steps N to reach the exit. This is the case because (1) there is only one exit state defining each sub-MDP, (2) that state is reachable with probability one, and (3) terminating the sub-MDP is constrained via this one exit state. This is the key property that makes it possible to additionally abstract a whole region into an aggregate state.

Independence means that $P^\pi(s', N|s, a) = P^\pi(s'|s, a).P^\pi(N|s, a)$. Equation 1 using abbreviation $a = \pi(s)$ becomes

$$V_m^\pi(s) = V_{m-1}^\pi(s) + \sum_N P^\pi(N|s,a)\gamma^{N-1} \times \sum_{s'} P^\pi(s'|s,a)[R + \gamma V_m^\pi(s')] \quad (2)$$

The *exit value function* E [7] for all states s in region g is the expected value of future rewards *after completing* the execution of the abstract action a exiting g and following the policy π thereafter.

$$E_m^\pi(g,a) = \sum_{s'} P^\pi(s'|s,a)[R + \gamma V_m^\pi(s')] \quad (\forall s \in g) \quad (3)$$

Definition 1 (Discount Function) *The* discount function D_m^π *is the expected discount to be applied to the exit value of* m *under policy* π *where* N *represents the random number of steps to exit.*

$$D_m^\pi(s) = \sum_N P^\pi(N|s,\pi(s))\gamma^{N-1} \quad (4)$$

Equation 2 is now succinctly written as

$$V_m^\pi(s) = V_{m-1}^\pi(s) + D_{m-1}^\pi(s).E_m^\pi(g,a) \quad (5)$$

The discount function D is itself recursively represented.

$$D_m^\pi(s) = \gamma D_{m-1}^\pi(s)\Gamma_m^\pi(g,a) \quad (6)$$

where

Definition 2 (Γ Function) *The* action discount function Γ *for all states* s *in region* g *is the expected value of discount that is applied to state* s' *reached* after completing *the execution of the abstract action* a *exiting* g *and following the policy* π *thereafter.* Γ *is constructed with the exit value of sub-MDP* m *set to* 1 *and all rewards zero.*

$$\Gamma_m^\pi(g,a) = \sum_{s'} P^\pi(s'|s,a)D_m^\pi(s') \quad (\forall s \in g) \quad (7)$$

were $P^\pi(s'|s,a)$ *is the probability of transitioning to state* s' *after abstract action* a *terminates from any state* $s \in g$*.*

Equations 3, 5, 6 and 7 are the decompositions equations for a discounted value function following policy π. The recursively optimal value and discount functions, where abstract action a invokes sub-MDP $m-1$ and state s is in region g, are

$$V_m^*(s) = \max_a [V_{m-1}^*(s) + D_{m-1}^*(s).E_m^*(g,a)] \quad (8)$$

$$D_m^*(s) = \gamma \max_a [D_{m-1}^*(s).\Gamma_m^*(g,a)] \quad (9)$$

This formulation requires only one value to be stored for functions E and Γ for all states in sub-MDPs for region g. In other words safe abstraction of the sub-MDP's states can be retained as in a non-discounted case. We achieve this at the cost of storing a separate on-policy action discount function. The overall benefit is that MDPs using cumulative discounted rewards can, in the best case, still scale linearly in space complexity in the number of variables.

Figure 1 is a simple illustration of how the overall value function is composed for a MDP with discounting. The key point is that we can compose the value function of the original MDP exactly by only storing the exit value function E for the two y labels at the top level sub-MDP (i.e. values 7.71 and 20.0).

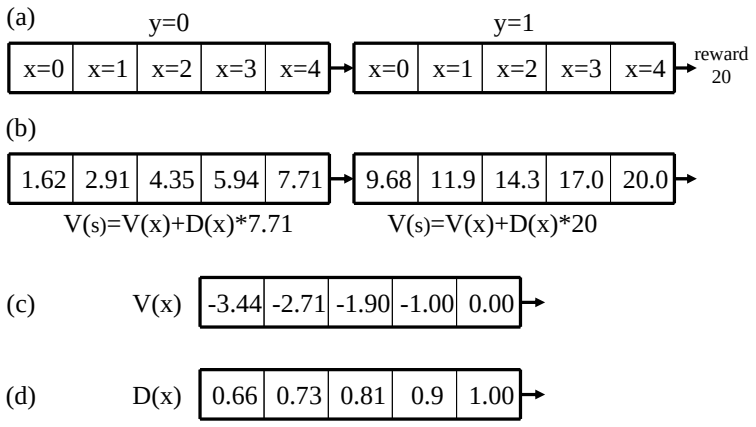

Figure 1: A simple example showing the state abstracted decomposition of a discounted value function. (a) shows a factored state MDP with two identical regions with one exit to the right. The deterministic actions are move left and right, all rewards are -1. The reward on termination is 20. The discount factor is 0.9. (b) the composed value function for each state. (c) and (d) are the abstracted sub-MDP value and discount functions respectively.

3 Infinite Horizon MDPs

The action value and discount function working together can decompose the original value function for all episodic policies. To allow a recursively optimal solution to be found for any HEXQ partitioned finite MDP we need to guarantee termination of sub-MDPs in the presence of positive and negative rewards. Even when all the rewards are negative an optimal policy does not not guarantee that the sub-MDP will exit when using a discounted value function.

We therefore define a pseudo value function, similar to that used in MAXQ, but one that has a large enough positive termination value to force any sub-MDP to exit. In MAXQ the pseudo reward value function sets large negative values on undesirable terminations. These are not required in HEXQ because the HEXQ partition and construction ensures unwanted exits cannot occur. Pseudo value functions can create sub-optimal policies in both MAXQ and HEXQ, but this is the price to pay for reusability of a manageable sub-task policy cache.

In summary, three decomposed value functions are required for each sub-MDP or sub-

task. The pseudo reward exit value function $\overline{E}$ determines the policies available as abstract actions at the next level up in the hierarchy. The function Γ holds discount values to be applied to the real exit value function at the next level up. The real exit value function E holds (in Ditterich's words) the "uncontaminated" exit values for each sub-MDP. $\overline{E}$ is updated on-line and Γ and E are simultaneously updated following $\overline{E}$'s policy.

A recursively optimal policy may require a sub-MDP never to exit. To allow for this possibility, we create an additional sub-MDP without exits for each region class in the HEXQ partition. This creates an abstract action at the next level and an extra policy in the cache that continues in the sub-MDP forever. To ensure that such a policy exists we need to ensure that regions are not forced to exit. HEXQ uses strongly connected components (SCC) to automatically construct regions. It is therefore a simple matter to meet this requirement. Any SCC with more than two states can have a no-exit (improper) policy. In the event of single state regions we only generate a no-exit policy if that state can transition to itself with probability one.

During learning a timeout is now required for sub-MDPs. A non-exiting sub-MDP will not return control to its parent and a means of interrupting the execution of the sub-MDP is required. We count the number of steps that a sub-MDP executes and if it exceeds a threshold value the execution is interrupted and control is passed back up the hierarchy to the top level without updating exit value functions.

4 Empirical Experiments

The following two experiment are designed to provide empirical evidence that HEXQ is able to perform in an infinite horizon setting, continue in a sub-task if required and successfully solve otherwise intractable problems. The HEXQ HRL algorithm [7] has been modified to include the new discount function and non-exiting sub-MDPs as outlined above.

Continuing Taxi. A taxi task, introduced by Dietterich [3] to demonstrate MAXQ function decomposition, is shown in figure 2. A taxi trans-navigates a grid world to pick up and drop off a passenger at four possible locations, designated R, G, Y and B. Actions move the taxi one square to the north, south, east or west, pickup or putdown the passenger. Navigation actions have a 20% chance of slipping to either the right or left of the intended direction. Generally all actions incur a reward of -1. If the taxi executes a pickup action at a location without the passenger or a putdown action at the wrong destination it receives a reward of -10. To make the task continuing, another passenger location and destination are created at random after the passenger is delivered and the taxi is mysteriously tele-ported to a random grid location. The reward for delivering the passenger is 200.

The taxi domain is also augmented with another source of income for the taxi. If the taxi transitions from below to the grid location marked with the $ sign in figure 2, the reward is a positive number. This may be interpreted as a local delivery run with guaranteed work but at a different rate of pay. The taxi problem can be modelled as a MDP with a two dimensional state vector $s = (\text{taxi location, passenger source/destination})$. There are 25 possible taxi locations and 20 possible pickup-destination combinations.

This problem has two optimal solutions depending on the value of the $ reward. If the $ is low, the solution is to continue to pick up and deliver passengers as per the original taxi problem. For larger $ values the taxi prefers to continue local delivery runs and ignore the passenger. As the local $ reward is the same in the context of any passenger pickup and destination location, HEXQ discovers one class of region for navigation. Performing local deliveries means deciding to continue in a navigation sub-task.

It is instructive to vary the local reward at location $ to see when the taxi decides to perform local deliveries instead of picking up and dropping off passengers. The local reward is

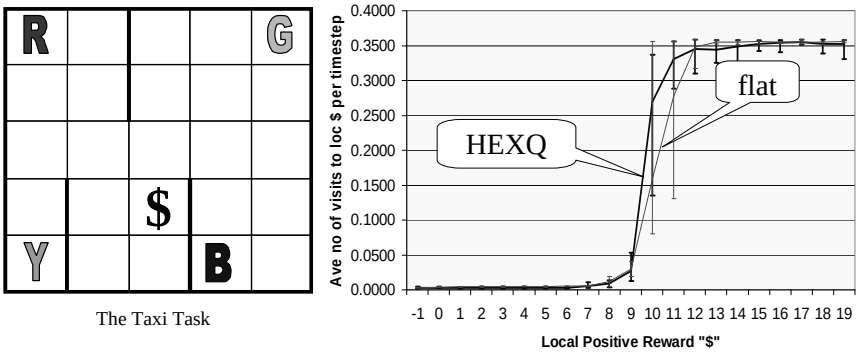

Figure 2: The infinite horizon taxi task. The graph shows that the HRL finds and switches policies just like the flat RL for various values of positive reward, confirming correct operation of the discount and value functions using state abstraction and non-terminating sub-tasks.

varied from -1 to 19 in increments of 1 and the MDP solved both with HEXQ and with a flat RL. The number of times the $\$$ location is visited per time step is counted as an indication of which strategy the taxi uses. A high visitation rate indicates the taxi prefers the local delivery run.

A HEXQ automatic decomposition of the original episodic taxi task creates 4 sub-MDPs at level 1 [7]. For the continuing taxi task we still only have 4 exit states at level 1, but an extra non-exiting sub-MDP is created as explained previously. Figure 2 indicates the optimal policy chosen for each value of local reward $\$$. As the amount of local reward increases the taxi switches strategy from delivering passengers to local delivery runs. The switch takes place when the local reward is about 10 for both HEXQ and the flat learner. The error bars indicate the maxima and minima over ten runs for each reward setting.

Within error bounds, both learners find similar solutions for this problem, providing confirming evidence that HEXQ can find the correct solution by persisting in a sub-task if necessary.

Soccer Player. The second example is a stylized bipedal robot that learns to walk, kick a ball and score goals. This problem is intractable on present day desktop computers requiring nearly 3 billion Q values. The experiment is designed to demonstrate the benefit of state abstraction in a discounted setting.

The MDP state description contains three variables: the robot leg stance and direction (384 labels), the position of the robot relative to the ball (861 labels) and the position of the ball on a soccer field (400 labels). 21 primitive actions allow the robot to move the legs and change its direction. The robot leg positions, soccer field and ball behaviour is shown in figure 3 (a) and (b). The reward on each state transition is -1 and a trial terminates when a goal is scored. When the robot runs into the ball, the ball is kicked stochastically but roughly in the direction the robot is moving. HEXQ automatically generates a 3 level hierarchy. A fuller description is beyond the scope of this paper. Our primary interest is to test HEXQ when positive reward is introduced. We separately use positive rewards to (1) reward the robot for running on the spot and (2) reward the robot for running around the ball. The significance of these conditions is that recursively optimal solutions can only be found by the robot continuing in a level 1 sub-task and level 2 sub-task respectively.

Using a learning rate of 0.25, a discount rate of 0.99 and an $\epsilon-greedy$ exploration policy for the top level sub-MDP with $\epsilon = .5$ for the first 500 trials, HEXQ generates a total of

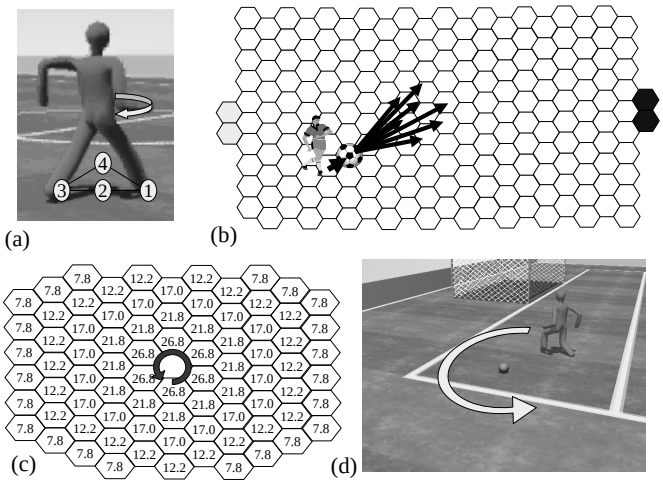

Figure 3: The soccer player showing (a) the simulated robot leg positions, (b) 400 discrete ball locations on the field, (c) the discounted value of states in the level 2 no-exit sub-MDP when the robot is rewarded for running around the ball and (d) a snapshot of the robot running around the ball.

17 sub-MDPs to decompose the problem. At level 1, 6 sub-MDPs determine the one step walk directions plus one no-exit sub-MDP. At level 2 there are two regions, one with 860 states and the other with one state when the agent is at the ball location. The first region has 7 sub-MDPs representing the 6 directions from which to approach the ball and one no-exit MDP. The single state region has two vestigial sub-MDPs, one allowing abstract actions to kick in one of six directions and a no-exit policy. After exploring the value function for scoring goals, the agent decides to continue at the no-exit level 1 sub-MDP running on the spot for case (1). For case (2) the agent continues in the level 2 no-exit sub-MDP for the larger region, running around the ball as shown in figure 3 (d). In this case it invokes terminating sub-MDPs from level 1 for locomotion. Figure 3 (c) shows the discounted value function for the no-exit sub-MDP for the larger region at level 2. Six maximum value states determine the circling policy.

HEXQ solves this problem in seconds with over four orders of magnitude saving in space requirements as a result of state abstraction made possible with the additional action discount function discussed in this paper.

5 Discussion and Future Work

This paper introduced a decomposed *action discount function* to make safe state abstraction possible in HEXQ. Commenting on options, Sutton, et al [9] say "The integration with state abstraction remain open and unclear". The ideas presented in this paper could easily be adapted to work with Options, MAXQ and HAMs under the constraints of a HEXQ partition.

A HEXQ partition allows stochastic exits. The more difficult condition to meet in HEXQ decomposing a MDP is that each exit state must be reachable from within a region class with probability one. Work in progress [10] to relax this constraint includes extending bounded parameter MDPs [11] to a semi-Markov setting using approximate HEXQ parti-

tion conditions or using overlapping regions as for “Airport Hierarchies” [12].

To reduce computational complexity hierarchical reinforcement learners generally constrain policies and therefore cannot make global optimality guarantees. For deterministic problems, HEXQ will find globally optimal solutions. In general, for stochastic problems, HEXQ can only find recursively optimal solutions based on the constructed hierarchy. In practice HEXQ finds good policies or even globally optimal policies for a range of simple stochastic problems with hierarchical greedy execution after learning.

References

[1] Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. *Advances in Neural Information Processing Systems 5 (NIPS)*, 1992.

[2] Leslie Pack Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Machine Learning Proceedings of the Tenth International Conference*, pages 167–173, San Mateo, CA, 1993. Morgan Kaufmann.

[3] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[4] David Andre and Stuart J. Russell. State abstraction for programmable reinforcement learning agents. In Rina Dechter, Michael Kearns, and Rich Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pages 119–125. AAAI Press, 2002.

[5] Doina Precup. *Temporal Abstraction in Reinforcement Learning*. PhD thesis, Univeristy of Massachusetts, Amherst, 2000.

[6] Ronald E. Parr. *Hierarchical Control and learning for Markov decision processes*. PhD thesis, University of California at Berkeley, 1998.

[7] Bernhard Hengst. Discovering hierarchy in reinforcement learning with HEXQ. In Claude Sammut and Achim Hoffmann, editors, *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 243–250. Morgan-Kaufman, 2002.

[8] Thomas Dean and Robert Givan. Model minimization in markov decision processes. In *AAAI/IAAI*, pages 106–111, 1997.

[9] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[10] Bernhard Hengst. Variable resolution in hierarchical RL. Technical Report UNSW CSE TR 0309, National ICT Australia, School of Computer Science and Engineering, University of New South Wales, Sydney NSW Australia, May 2003.

[11] Robert Givan, Sonia M. Leach, and Thomas Dean. Bounded-parameter markov decision processes. *Artificial Intelligence*, 122(1-2):71–109, 2000.

[12] Andrew Moore, Leemon Baird, and Leslie Pack Kaelbling. Multi-value-functions: Efficient automatic action hierarchies for multiple goal mdps. In *Proceedings of the International Joint Conference on Artificial Intelligence, Stockholm*, pages 1316–1323, 340 Pine Street, 6th Fl., San Francisco, CA 94104, 1999. Morgan Kaufmann.