Termination Approximation: Continuous State Decomposition for Hierarchical Reinforcement Learning

Sean Harris, Bernhard Hengst, Maurice Pagnucco
UNSW Australia, NSW 2055, Australia

Abstract

This paper presents a divide-and-conquer decomposition for solving continuous state reinforcement learning problems. The contribution lies in a method for stitching together continuous state subtasks in a near-seamless manner along wide continuous boundaries. We introduce the concept of *Termination Approximation* where the set of subtask termination states are covered by goal sets to generate a set of subtask option policies. The approach employs hierarchical reinforcement learning methods and exploits any underlying repetition in continuous problems to allow reuse of the option policies both within a problem and across related problems. The approach is illustrated using a series of challenging racecar problems.

Introduction

Many real-world control problems are naturally represented using continuous state variables. This problem class includes the control of dynamic systems such as vehicles, robots, and plants, in a variety of engineering applications. Examples include 2D robotic navigation tasks and bipedal locomotion for a highly articulated robot. This class of problems can be solved using Reinforcement Learning (Barto and Sutton 1998).

Many real systems are also characterised by repetition in the dynamics of their sub-systems. From a reinforcement learning viewpoint, this means the transition and reward functions are similar in several different parts of the state-action space. The similarity can arise from symmetry, repetitive motor motions, reoccurring environmental conditions, or partial independence of system function. An inverted pendulum, for example, has similar left-right behaviour with the axis of symmetry of the pendulum system upright and at rest. The dynamics of a cart also remain the same regardless of its position on a table. A racetrack may be divided into several straight or similarly curved sections of road, where even left and right curved sections show symmetry.

It is known that Hierarchical Reinforcement Learning (HRL) can exploit repetition in systems, and can significantly reduce the computational complexity of finding good solutions (Barto and Mahadevan 2003). To date, HRL methods have primarily been applied to systems that constrain the continuous variables to whole sub-systems or concatenate continuous spaces with very narrow "doorways" (Mahajan 2014). As a result, there are a range of problems that contain high levels of repetition, but are not currently suited to HRL approaches because they contain wide boundaries between repetitive subtasks. In addition to this, similarities across related problems are rarely exploited for policy reuse, resulting in policies being relearned rather than transferred from prior learning.

Copyright © 2015, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

The main contribution of this paper is the decomposition and reconstitution of continuous state problems by partitioning the state space along wide continuous boundaries. This improves and expands the application of HRL to continuous control problems and provides a framework for transferring subtask policies between related problems. First, a problem is manually partitioned into subtasks to capture any repetition. Then the *termination set* between subtasks is approximated with local policies that form a set of abstract actions. This allows the subtasks to be recombined with an abstract MDP. We rely on relativised options (Ravindran and Barto 2003) and MAXQ value function decomposition (Dietterich 2000). We focus our research on episodic (terminating) tasks with an undiscounted value function.

The core challenge of this paper is to stitch together neighbouring continuous state spaces along their common boundary, which may be arbitrarily wide. We introduce *Termination Approximation* to approximate all possible ways to terminate a subtask. Termination Approximation defines a set of abstract goals that cover continuous states reachable after terminating the subtask. Each abstract goal defines a separate Markov option that is learned using a pseudo reward function to terminate the option. The number of goals covering the termination area directly determines how seamlessly the value function tracks across the boundary.

We present results from a simulated racecar problem. The task contains a high level of repetition, but is challenging to decompose effectively using current HRL approaches due to its wide continuous boundaries.

Related Work

HRL has been applied to many discrete problems and a limited number of continuous state problems. This paper extends HRL approaches to decompose increasingly challenging continuous state problems and allow policies to be

shared amongst related problems.

Hierarchical reinforcement learning has been well studied on simple, discrete domains. Research into this area began with 3 primary approaches, options, MAXQ and HAMs (Barto and Mahadevan 2003). The options framework (Sutton, Precup, and Singh 1999) integrated temporally extended actions (eg. macro actions) into the reinforcement learning framework, improving the speed and robustness of RL systems. MAXQ (Dietterich 2000) showed that a problem could be decomposed into simpler subtasks, solved individually and recombined to solve the original problem. MAXQ also utilised state abstraction, where irrelevant parts of the state were excluded for particular problems, substantially reducing the learning requirements. State abstraction allows learning to occur in a more general context and thus is a particularly important concept for transfer learning.

The options framework has been applied to continuous state problems through skills chaining (Konidaris and Barreto 2009). This allows temporally extended actions to be utilised in a continuous state space, but is focused on autonomous generation and does not integrate state abstraction. Even where there is underlying repetition in a problem, policies are relearned for each subtask. Policies are also not shared between related tasks. Relativised options (Ravindran and Barto 2003) introduced the concept of state abstraction to the options framework, but is only compatible with discrete, highly structured environments that possess "funnel-states". Although some environments, such as buildings with rooms and doorways, contain features that create natural funnel-states, the general class of RL problems does not. Reliance on environmental features to create funnel states and decompose a problem also reduces the ability to transfer learning to a related task where those same environmental features may not be present, or may be present but in a different manner. This paper extends relativised options for continuous state spaces as part of a larger hierarchical approach, allowing the use of state abstraction with the options framework on a series of related continuous state problems.

MAXQ has also been extended to solve problems with continuous subtasks through Fitted R-MAXQ (Jong 2010). Whilst this work utilises state abstraction effectively to solve problems with continuous subtasks, the continuous subtasks themselves are not decomposed, thus limiting the applicability of this approach to continuous state problems. This work provides a framework to decompose and recombine continuous subtasks, thus expanding the current state of the art.

This paper extends MAXQ in combination with relativised options to allow the decomposition of continuous state problems into subtasks and apply state abstraction to minimise the learning requirements across subtasks and related problems. The main contributing feature of the decomposition is that subtasks can have arbitrary continuous boundaries. This allows a problem to be decomposed around repetition of subtasks, instead of environmental features, improving the ability to transfer learning between related tasks. We introduce the concept of *Termination Approximation* to generate subtask option policies.

Background

The problems being examined in this paper can be modelled as a *Markov Decision Process* (MDP) $\langle S, A, \Psi, P, R\rangle$ (Ravindran and Barto 2003) where S is a continuous set of states, A is a discrete set of actions, $\Psi \subseteq S \times A$ is the set of admissible state-action pairs, $P : \Psi \times S \to [0, 1]$ is the probabilistic transition where the probability of transitioning from state s to state s' under action a is $P(s, a, s')$, $R : \Psi \to \mathbb{R}$: the reward function where the reward for performing action a in state s is $R(s, a)$. Let $A_s = \{a|(s, a) \in \Psi\} \subseteq A$ be the set of actions admissible in state s. We assume that each state has an action available in it, i.e. $\forall s \in S, A_s \neq \varnothing$

To solve an MDP we learn a *policy* $\pi : \Psi \to [0, 1]$ which is the probabilty of taking action a in s for all $(s, a) \in \Psi$. The solution to an MDP is an *optimal policy* π^*, which uniformly dominates all other policies for that MDP. The *action-value function* $Q^\pi : \Psi \to \mathbb{R}$ maps state-action pairs (s, a) to their expected sum of future rewards for performing action a in state s then following π thereafter.

An option is defined by the tuple $O = \{I, \pi, \beta\}$, where the initiation set $I \subseteq S$ is the set of states the option can be invoked, $\pi : \Psi \to [0, 1]$ is the policy and the termination set $\beta :\to [0, 1]$ is the probability of the option terminating in each state. In this paper we restrict our focus to Markov options.

Relativised options and the MAXQ value function decomposition are both utilised in this paper. Here we examine the theory behind them.

Relativised Options

Options (Sutton, Precup, and Singh 1999) are extended to form relativised options (Ravindran and Barto 2003) by removing the absolute frame of reference. The option is learned in a reduced MDP M' and mapped onto a variety of subtasks in the original problem where the local environment is symmetric or repetitive. An *MDP homomorphism* from M to M' is introduced to map states in M to equivalent states in M'

Definition: An *MDP homomorphism* h from an MDP $M = \langle S, A, \Psi, P, R\rangle$ to an MDP $M' = \langle S', A', \Psi', P', R'\rangle$ is a surjection from Ψ to Ψ', defined by a tuple of surjections $\langle f, \{g_s|s \in S\}\rangle$, with $h((s, a)) = (f(s), g_s(a))$, where $f : S \to S'$ and $g_s : A_s \to A'_{f(s)}$ for $s \in S$ such that $\forall s, s' \in S, a \in A_s$:

$$P'(f(s), g_s(a), f(s')) = \sum_{s'' \in [s']_f} P(s, a, s'') \qquad (1)$$

$$R'(f(s), g_s(a)) = R(s, a). \qquad (2)$$

The surjection f maps states of M to states of M' whilst each surjection g_s recodes the actions admissible in state s of M to actions admissible in state $f(s)$ of M'. Condition (1) says that the homomorphism commutes with the system dynamics whilst condition (2) says that state-action pairs with the same image under h have the same reward. A policy in M' is said to *induce* a policy in M. This leads to the definition of *relativised options*, where we map our original MDP M to an option MDP M_O.

Definition: A *relativised option* of an MDP $M = \langle S, A, \Psi, P, R\rangle$ is the tuple $O =< h, M_O, I, \beta >$, where $I \subset S$ is the initiation set, $\beta : S' \rightarrow [0, 1]$ is the termination function and $h = \langle f, \{g_s | s \in S\}\rangle$ is a partial homomorphism from the MDP $\langle S, A, \Psi, P, R_O\rangle$ to the option MDP $M_O = \langle S', A', \Psi', P', R'\rangle$ with R_O chosen based on the subtask.

MAXQ

MAXQ (Dietterich 2000) provides a formal framework for using a hierarchy of subtasks to decompose a problem. It allows for state abstraction within subtasks as well as the decomposition of the value function across subtasks. The expected reward is decomposed by dividing the value function recursively until it reaches the primitive actions. It introduces a completion function $C(i, s, a)$, which is the expected discounted cumulative reward of completing subtask i after completing the subtask a. Formally this is defined as:

$$C^\pi(i, s, a) = \sum_{s',N} P_i^\pi(s', N|s, a)\gamma^N Q^\pi(i, s', \pi(s')) \quad (3)$$

In equation 3, N refers to the number of time steps taken to complete an action and γ is the discounting factor used to discount future rewards. This equation allows the action-value function to then be expressed recursively:

$$Q^\pi(i, s, a) = V^\pi(a, s) + C^\pi(i, s, a) \quad (4)$$

V^π is the cost of the current abstract action, which in MAXQ can be composed of various smaller subtasks. V^π is evaluated recursively until it reaches a primitive action. Ignoring the details of how V^π is calculated, the action-value function is the cost of the current abstract action plus the cost to goal following policy π afterwards. This ability to decompose the value function allows the expected cost to goal to be calculated across a hierarchy, thus giving us close to the same representational power as a flat learner, but with the time and storage savings of a hierarchy.

Continuous State Decomposition

Motivating Example

We illustrate our decomposition of continuous state spaces with a challenging racecar problem. The problem task is to drive a racecar around one lap of a simulated racetrack. The state space is continuous and four dimensional, $S = \{x, y, heading, velocity\}$ and the action space has 9 discrete actions. The actions allow the agent to vary the $heading$ and $velocity$ of the car simultaneously and independently. Both variables can be increased a small amount, remain the same, or decreased a small amount at each time step. Movements are deterministic; at each time step the agent moves $velocity$ metres in the $heading$ direction. One example racetrack used for testing was approximately 63m long and is shown in Figure 1.

This problem is challenging because the state space is continuous, not a grid world, and there are no natural doorways or funnel states to provide simple decomposition. Despite these challenges, the problem shows potential for decomposition, policy reuse and state abstraction. The racetrack is composed of only straight and turn regions, the dynamics of which remain the same no matter where on the racetrack they lie. Different racetracks also contain the same fundamental building blocks of straight and turn regions. This means that if we were able to learn policies to navigate straight and turn regions of track, there is potential to decompose the task and reduce the overall learning requirements.

The rest of this section decomposes the racecar example incrementally to introduce our approach to continuous state decomposition. We begin by examining a decomposition of the problem and how to solve each subtask using options, introducing *Termination Approximation*. We then highlight the ability to state abstract each subtask to vastly reduce learning time and finally form an abstract MDP to solve the overall problem.

Decomposition

We utilise a *region-based decomposition* of a given MDP as defined in (Hauskrecht et al. 1998).

Definition: A region-based decomposition Π of an MDP $M = \langle S, A, T, R\rangle$ is a partitioning $\Pi = \{S_1, ...S_n\}$ of the state space S. The elements S_i of Π are called *regions* of M.

Associated with each region, we introduce a *termination set* of states.

Definition: The *termination set* of states $TS_{S_i} = \{t \in S - S_i : T(s, a, t) > 0$ for some a, $s \in S_i\}$ contains all states not in S_i that are reachable with probability greater than zero after executing a primitive action from $s \in S$.

Definition: The combination of the region states S_i and the termination set TS_{S_i} is represented by $X_i = S_i \cup TS_{S_i}$.

This allows us to define a *region option* to navigate from a region to its termination set.

Definition: A region option contains three components, a policy $\pi : \Psi \rightarrow [0, 1]$, a termination condition $\beta(s) = \begin{cases} 1 & : s \in TS_{S_i} \\ 0 & : s \notin TS_{S_i} \end{cases}$ and an initiation set $I \subseteq S_i$. Note that $\Psi = X_i \times A$, meaning that the option is defined across both the region S_i and its termination set TS_{S_i}. The termination function is always false inside the region, but always true inside the termination set.

Since the termination set is comprised of states outside of S_i, and the region-based decomposition partitions the entire state space into regions, it follows that the termination set TS_{S_i} for a region S_i is comprised of states from other regions. This "overlap" means that if we follow a region option, it will take us from inside the region, into that region's termination set, which is inside another region. From this region we can follow a new region option, and thus chain together region options to complete the overall task. This chaining approach is similar to that used in Skills Chaining (Konidaris and Barreto 2009). An example decomposition of the racecar problem is shown in Figure 1. The racetrack is decomposed into a series of straight and turn regions, separated by the red lines.

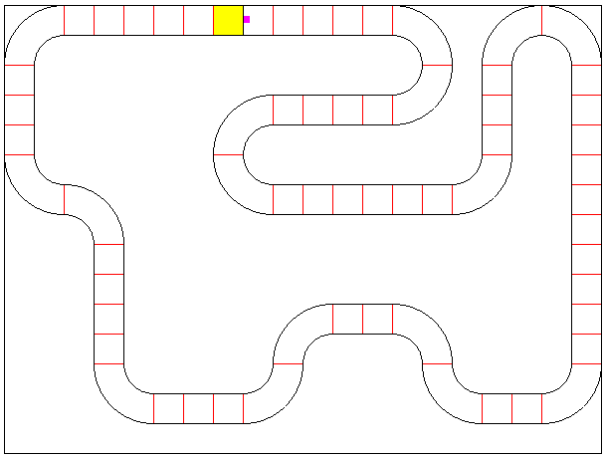

Figure 1: An example racetrack decomposed into regions. The agent starts at the small purple dot and at the top and must drive a full lap and enter the yellow goal region to finish

Termination Approximation

One problem with this decomposition is that many problems, including the racacar problem, contain a wide continuous boundary between regions. Although only learning one region option per region will give a solution to the overall problem, the solution may be suboptimal. The primary restriction on optimality is that the wide boundary results in a lack of control over *how* the agent transitions between regions. We introduce *Termination Approximation* as a method for solving this problem. It involves learning multiple region options per region and utilising an abstract MDP to learn which region option is best for each region. Before we examine this in more detail, let us first introduce the abstract MDP in more detail.

As part of the region based decomposition, Hauskrecht et al propose forming an abstract MDP to hierarchically solve problems (Hauskrecht et al. 1998). Each region becomes an abstract state, whilst the option policies learnt for each region become the abstract actions. Hauskrecht utilised one region option per termination state, which for a discrete problem with funnel-states was often a very low number, and thus were able to efficiently generate enough low level policies to solve the problem.

Utilising this hierarchical approach on continuous state problems without funnel-states provides extra challenges, primarily because attempting to learn one option policy for every state in the termination set would be more expensive than solving the original MDP. Instead we *approximate* this set of policies and only find a near-optimal solution.

Our method for approximating this set of policies is to uniformly partition the termination set into a series of goal sets and learn one option policy per goal set. We make the assumption that options terminating in similar subsets of the termination set are very similar and thus can be approximated by a single policy. The coarseness of the partitioning therefore directly influences how well this set of policies approximates the real set of policies. We also suggest that although a hand coded partitioning of the termination set for a specific problem could produce a more efficient solution, a uniform partitioning is adequate for most applications.

Learning region option policies and ensuring they terminate in the desired goal set is achieved using pseudo-rewards. If the region option terminates outside its particular goal set, the agent receives an extra (pseudo) reward to discourage such termination in the future. This allows each option policy to be shaped such that it learns to terminate in the correct goal set. The value function is polluted by such pseudo-rewards though, making it unusable for learning a MAXQ completion value (Dietterich 2000). This is solvable by learning a second value function that isn't polluted to represent the expected reward without contamination.

Relativising Regions and Options

The next aspect to consider is that the dynamics of some regions may be repeated in the problem. In the racecar problem we can see that the straight regions have the same dynamics no matter where in the racetrack they lie. The same also applies to the turn regions. Thus there is a significant potential for additional state abstraction by sharing policies between regions.

State abstraction has already been applied to options through *relativised options*. Relativised options are learnt in a relative state without an absolute frame of reference. They are reused through homomorphisms mapping the original MDP to the option MDP. We mimic this approach to define *relativised region options*.

Definition: A *relativised region option* of an MDP $M = \langle S, A, \Psi, P, R \rangle$ is the tuple $O =< h, M_O, I, \beta >$, where $I \subset S$ is the initiation set, $\beta(s) = \begin{cases} 1 & : s \in TS_{S_i} \\ 0 & : s \notin TS_{S_i} \end{cases}$ is the termination function and $h = \langle f, \{g_s | s \in S\} \rangle$ is a partial homomorphism from the MDP $\langle S, A, \Psi, P, R_O \rangle$ to the option MDP $M_O = \langle X', A', \Psi', P', R' \rangle$ with R_O chosen based on the subtask. Note that $\Psi' = X' \times A'$ since the relativised region options is defined over the relativised region and its termination set.

Thus the option MDP M_O is a partial homomorphic image of an MDP with the same states, actions and dynamics as M but with its own reward function specialised for the subtask. The policy $\pi : \Psi' \rightarrow [0, 1]$ is obtained by solving M_O treating it as an episodic task. This is the same definition as a relativised option except that the MDP and termination function are defined across X', the union of the region states S' and the termination set $TS_{S'}$ instead of over just S'. This follows from our previous approach of learning a region option that terminates in the termination set, just in a relativised format.

Applying this to the racecar problem, we now only need to learn two sets of relativised region options to be able to complete the track, one set for a straight region and one set for a turn region. Each relativised region option can then be reused through the homomorphisms to allow the agent to solve a repeated instance of a region without having to relearn any region options.

Abstract MDP

The solution to the abstract MDP is found by learning a MAXQ completion function. The formula for this is shown in Equation (5). It represents the expected cost to goal after completing the current abstract action (relativised region option). It is slightly modified from Equation (3) as we do not use the N or γ terms due to the episodic nature of the problems we are solving. In fact, we cannot use discounting in our approach currently or state abstraction is not possible (Dietterich 2000). This means that continuing problems are not yet solvable by this method. Note that the formula for Q is unchanged from (4). The algorithm for learning the top level MDP is shown in Algorithm 1

$$C^{\pi}(i,s,a) = \sum_{s'} P_i^{\pi}(s'|s,a) Q^{\pi}(i,s',\pi(s')) \qquad (5)$$

Algorithm 1 Learning a policy for the abstract MDP

while $learning == true$ **do**
  $agent \leftarrow startingstate$
  while $atGoal() == false$ **do**
    $s = currentAbstractState$
    choose an action a according to the current exploration policy $\pi(s)$
    let π_a be the option policy corresponding to action a.
    "take action" - follow π_a until termination
    observe result abstract state s'
    observe starting option state s_o
    $C_{t+1}(s) \leftarrow (1-\alpha) \cdot C_t(s) + \alpha(V_t(s_o) + C_t(s'))$
  end while
end while

Transfer Learning

This decomposition approach allows a problem to be decomposed into small, fundamental regions that are readily repeated within a problem. This contrasts against previous decomposition methods which focus around natural or environment funnel-states within a problem. Since the decomposition is no longer focused on the environment, but the problem dynamics, transferring policies between related problems becomes more achievable. Any problem with the same or similar dynamics can benefit from the learning in another problem.

Let us examine the racecar problem again. To learn to drive the track in Figure 1, we learn a set of policies for a straight region and a set of policies for a turn region, then solve an abstract MDP. Now if we come across a new racetrack that looks nothing like our current track, we can take the same approach and decompose it into straight and turn regions. We can then use the set of policies learned for the previous racetrack's straight and turn regions again on this racetrack, and only have to learn the abstract MDP. The abstract MDP takes significantly less time to solve than a regular flat learner would and thus we are able to save significant amounts of time by sharing learning across related problems.

Results

Implementation Details

Our approach to learning in the continuous state space at the low level of the racecar problem utilises QLearning with uniform sampling of the state space and linear interpolation to approximate the value function between samples. We sample every $0.05m$ across the x and y dimensions, every 10^o across the $direction$ dimension and every $0.1ms^{-1}$ across the $velocity$ dimension. The reward function is a cost of 1 per action taken. Each straight region is $1m \times 1m$ in x and y and includes all possible values for the other dimensions. Each turn region fits in a $2m \times 2m$ box in x and y and turns 90^o. It also includes all possible values for the $velocity$ and $direction$ dimensions. The range of the velocity permitted is $0-0.4m/s$. The action space allows the racecar to change its heading by 10^o, 0^o or -10^o and the velocity by $0.1m/s$, $0m/s$ or $-0.1m/s$.

We approximate the termination area with 5 goals in each dimension across y, $direction$ and 4 across the $velocity$.

The method for learning the low level policies is not significant to the final result. We use QLearning due to its wide applicability to a variety of domains and linear interpolation since it is simple to scale efficiently to a large number of dimensions, but the policy could instead be learnt by other RL approaches, by demonstration, or even provided by the programmer.

Transfer Learning

To consider the power of this approach for transfer learning, we assume that the low level policies are already learned and only the abstract MDP needs to be solved. We chose to partition the termination set into 100 goal sets, partitioning it into 5 blocks across y, 5 blocks across direction and 4 blocks across velocity. It is possible to use other partitions with more or less goal sets than this. We chose 100 because it was close to the maximum number of policies we could store in memory on our test PC. The policies were learned once in isolation and reused for each of the experiments. Learning the low level policies took approximately $86 * 10^3$ seconds in total.

We learned a variety of racetracks of different shapes and sizes and compare the results against a baseline learner. The baseline was the same setup that was used to learn the low level policies, using bilinear interpolation and Qlearning. For each different racetrack the HRL agent converges to a hierarchically optimal policy, which is suboptimal compared to the baseline. This is expected, since the hierarchical structure places artificial restrictions on the policy that can be learned. The HRL agent also learns to solve each racetrack significantly faster than the baseline learner, ranging between $15\times$ and $30\times$ faster. This is due to the fact that the abstract MDP is smaller and simpler than the original problem and thus the learning converges substantially faster.

Table 1 shows the baseline results while Table 2 shows the HRL results. We measured the number of timesteps the agent takes to complete the task to calculate a regret measure to compare the policies with. We also used CPU time to

measure how fast learning converged after a fixed number of episodes (9.5×10^6 for the baseline, 0.1×10^6 for the HRL).

Table 1: Timing and path lengths for the baseline learner on a series of racetracks

Racetrack	Steps	CPU Time (s)
baseline m1	20	0.60×10^3
baseline m2	75	3.35×10^3
baseline m3	100	5.91×10^3
baseline m4	211	24.22×10^3

Table 2: Timing, path lengths and comparisons for the HRL learner on a series of racetracks

Track	Steps	CPU Time (s)	Regret	Speed Up
hrl m1	23	0.06×10^3	13%	10×
hrl m2	94	0.35×10^3	20%	10×
hrl m3	122	0.49×10^3	18%	12×
hrl m4	261	0.80×10^3	19%	19×

Another point to notice is that the time savings increase as the size of the racetracks increases, showing how the abstract MDP is less vulnerable to the curse of dimensionality than the baseline learner.

Figure 2 shows a comparison between the baseline policy and HRL policy for the example racetrack (m4).

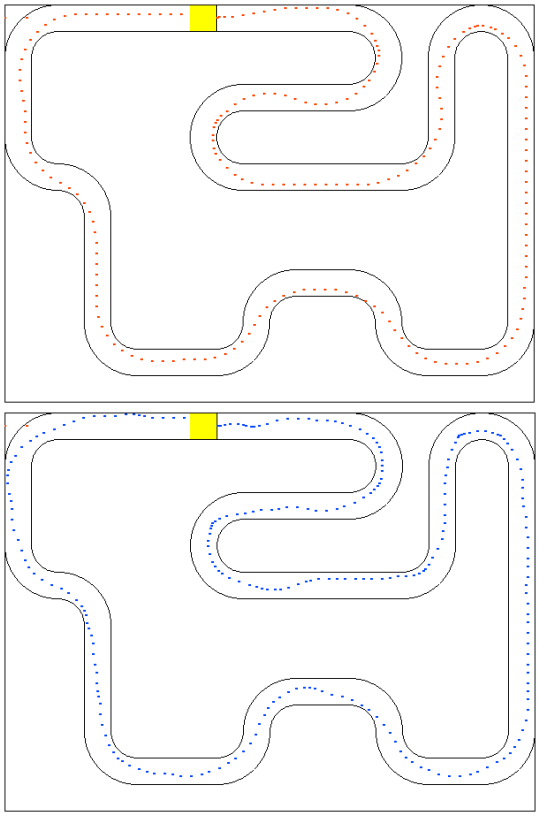

Figure 2: The baseline (orange) policy compared to the HRL (blue) policy.

Future Work

Our continuing work includes a more thorough evaluation of the effect of varying the coarseness of the termination set partitioning as well as improvements to the efficiency and range of problems that can be solved with our approach.

Coarseness Evaluation: A thorough evaluation of the coarseness of the termination set partitioning in comparison with the hierarchically optimal policy length will provide a good guide as to what coarseness is appropriate for different problems. Using 100 policies had a large initial learning overhead, whereas a smaller number of policies may give a large time speed up with only minor path length degradation.

Efficient Learning: The commonalities between each of the relativised options learnt for a region could be exploited. The airports hierarchy (Moore and Baird 1999) is a multi-goal based hierarchical data structure that minimises both the learning time and storage requirements for similar policies. It exploits the relationship between nearby goal states to guide the agent towards a similar goal from far away and only uses the final goal's policy when it is much closer. There is a very small reduction in optimality, but a large reduction in computational complexity. Incorporating the airports hierarchy into this work would allow a reduction in learning time per abstract goal learnt for each region.

Discounted Reward: At present solving problems with discounted reward is not possible since we do not know how many time steps a low level policy will take to complete and thus how much to discount(Dietterich 2000). It is possible to learn an additional discount function at the low level to solve this problem (Hengst 2007). The addition of this discount function would allow this approach to also be applied to continuing problems such as the pole and cart problem which are currently not solvable using this approach.

Conclusion

In this paper we have presented a novel method for decomposing continuous state spaces across wide boundaries. Our method allows continuous state problems to be decomposed based on repetition in the problem, not around environmental features, allowing for maximal reuse no only within a problem, but between related problems. We used termination approximation to address the challenge of stitching together wide boundary areas between regions and relativised options to allow maximal policy reuse. We demonstrated our approach on a challenging racecar problem that was previously not able to be effectively solved using HRL, highlighting how repetition can be utilised despite wide boundaries between regions.

References

Barto, A., and Mahadevan, S. 2003. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems* 13:341–379.

Barto, A., and Sutton, R. 1998. *Reinforcement learning: An introduction*.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research* 13(1):227–303.

Hauskrecht, M.; Meuleau, N.; Kaelbling, L. P.; Dean, T.; and Boutilier, C. 1998. Hierarchical Solution of Markov Decision Processes using Macro-actions. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*.

Hengst, B. 2007. Safe state abstraction and reusable continuing subtasks in hierarchical reinforcement learning. *AI 2007: Advances in Artificial Intelligence* 58–67.

Jong, N. K. 2010. *Structured Exploration for Reinforcement Learning*. Ph.D. Dissertation.

Konidaris, G., and Barreto, A. S. 2009. Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining. In Bengio, Y.; Schuurmans, D.; Lafferty, J. D.; Williams, C. K. I.; and Culotta, A., eds., *Advances in Neural Information Processing Systems 22*. Curran Associates, Inc. 1015–1023.

Mahajan, S. 2014. Hierarchical Reinforcement Learning in Complex Learning Problems: A Survey. *International Journal of Computer Science and Engine Science and Engineering* 2(5):72–78.

Moore, A., and Baird, L. 1999. Multi-value-functions: Efficient automatic action hierarchies for multiple goal MDPs. 1316–1323.

Ravindran, B., and Barto, A. 2003. Relativized options: Choosing the right transformation. In *ICML*, 608–615.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112(1-2):181–211.