A Core Task Abstraction Approach to Hierarchical Reinforcement Learning

(Extended Abstract)

Zhuoru Li[†], Akshay Narayan[†] and Tze-Yun Leong[†§]
School of Computing, National University of Singapore[†]
School of Information System, Singapore Management University[§]
{lizhuoru, anarayan, leongty}@comp.nus.edu.sg

ABSTRACT

We propose a new, *core task abstraction* (CTA) approach to learning the relevant transition functions in model-based hierarchical reinforcement learning. CTA exploits contextual independences of the state variables conditional on the task-specific actions; its promising performance is demonstrated through a set of benchmark problems.

Keywords

hierarchical reinforcement learning; core task abstraction

1. INTRODUCTION

In hierarchical reinforcement learning (HRL), a complex problem is solved by recursively decomposing the problem into smaller subtasks at different levels of details. Most model-based methods derived from the benchmark MAXQ framework [2] solve HRL problems by learning the individual task transition dynamics in a task hierarchy of the problem space. We propose a new learning algorithm that can efficiently compute the task transition functions in the hierarchy. For instance, Figure 1 shows the MAXQ task hierarchy for a taxi problem, where the objective for the agent is to pick up and deliver a passenger to the destination. The root problem is decomposed into the *Get* and *Put* tasks, which can be further decomposed into navigation subtasks and primitive actions. Recent improvements to MAXQ include integrating directed exploration [4], using Bayesian priors [1] and Monte-Carlo simulation [6]. One major problem of the MAXQ based methods is that many similar subtasks that potentially share the same transition dynamics are created. For example, in Figure 1 there are four navigation subtasks, one for each possible destination. These subtasks are independent, hence more exploration is needed to learn a good policy. Our approach aims to avoid the redundant work by reusing the common "knowledge".

We introduce the concept of *fragments* (partial structures shared across tasks) that supports simultaneously learning multiple task transition functions. For example, all the navigation subtasks in Figure 1 that share the same transition dynamics but differ in terminal states and local reward functions can be captured using a single navigation fragment. We also exploit the contextual independences of state variables conditional on the action executed, i.e. the state variables will have different sets of parents, depending on the action executed.

Our method assumes given information on safe state abstraction (i.e. ignoring the features that do not affect the values of the states), as in current MAXQ based methods. Based on our algorithm, however, an agent's experience in learning one task would help speed up learning all the other tasks with similar features. This leads to better empirical performance as compared with existing methods. Such abstraction information can be interactively or automatically learned in future.

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

2. CORE TASK ABSTRACTION

Consider the hierarchy in Figure 1. Each non-leaf node in this hierarchy is a task. A "task" is a unit of execution, similar to an option [5]. It has an input set I and terminating states G. It is a well-defined factored Markov decision process (MDP) representation of a single problem— it has a set of state variables or features F, a set of actions A, a transition function T and a reward function R. Here, F and A are given, T and R are to be learned.

DEFINITION 1. *A fragment j is a tuple $\langle F_j, A_j, T_j \rangle$ where F_j is the set of relevant features, A_j is the set of applicable actions and T_j is the local transition function.*

Unlike a task, a fragment does not have an input set, terminating states and a local reward function, and thus is not a well-defined MDP. A fragment corresponds to multiple similar tasks that share the same transition dynamics, but differ in the terminating states and local reward functions. Learning a fragment allows the agent to perform well in all the tasks by sharing the learned knowledge across tasks. We can redefine the MAXQ hierarchy in Figure 1 using fragments for navigation, which is common to both *Get* and *Put*. For simplicity of exposition, we use the term *node* to refer both task and fragment. We use N to denote the set of all non-root nodes and X to denote a subset of N.

To obtain the transition function using CTA, we identify the features unique to a subset of nodes under consideration. We define a unique function to do this as follows. $Uni(\bigcap_{i\in X} F_i) = \bigcap_{i\in X} F_i - \bigcup_{j\in \overline{X}} F_j$, where the first term is a set of features relevant to X, and the second term is not.

THEOREM 1. *Let $X_a = \{i | i \in N \wedge a \in A_i\}$ represent the set of nodes where action a is applicable. The transition*

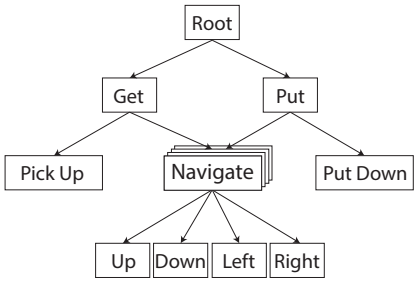

Figure 1: Task hierarchy for the taxi problem. There are four navigation subtasks, which can be represented using a single fragment.

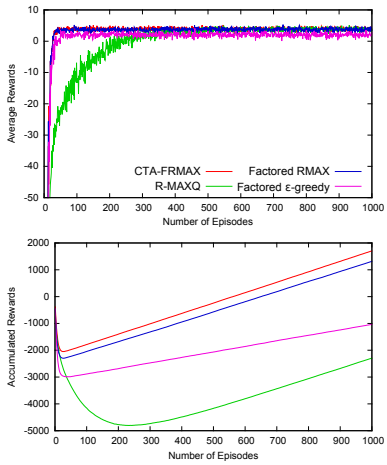

Figure 2: Results on MAXQ taxi.

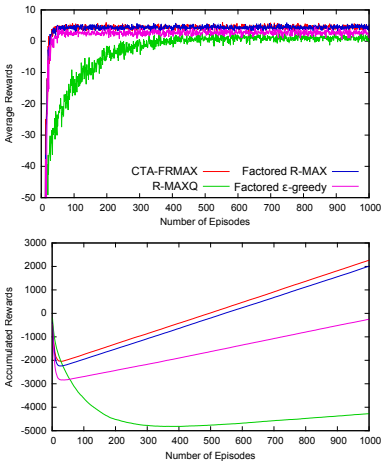

Figure 3: Results on fickle taxi.

model for node i is given by,

$$P(F_i'|F_i,a) = \prod_{Y:Y\subseteq N\wedge i\in Y} P\left(Uni(\bigcap_{j\in Y} F_j') \mid \bigcap_{j\in X_a\cap Y} F_j, a\right)$$

The theorem states that the transition function is a product of factors where, the parent set of $Uni(\bigcap_{j\in Y} F_j')$ depends on the action being executed. This is captured in the relation $\bigcap_{j\in X_a\cap Y} F_j$.

The direct outcome of the theorem is a reduced feature set for learning. We combine the safe state abstraction information and contextual independence to eliminate the maximum number of features from the parent set to improve the learning efficiency.

We introduce the CTA-FRMAX algorithm by combining efficient transition function learning in Theorem 1 with RMAX style exploration in factored spaces [3]. CTA-FRMAX maintains an exploration count on each factor in Theorem 1. If the count is smaller than a threshold, the agent transits to a fictitious state and receives maximum reward.

3. RESULTS

We test the performance of CTA-FRMAX in the original MAXQ taxi and the fickle taxi benchmark problems [2]. The experimental setup involves a 5×5 grid-world with four landmarks. The passenger location and destination must be one of the landmarks. There are six actions: *navigate* along the four directions, *pickup* and *putdown* as shown in Figure 1. In the fickle taxi variant the passenger may change the destination with probability 0.3. The episode ends if passenger reaches the destination or 1000 time steps elapse.

CTA-FRMAX computes the transition function for both *Get* and *Put*, and executes *Get* when the passenger is not on board and *Put* otherwise. Navigation is modeled as a fragment. We compare the performance of CTA-FRMAX against R-MAXQ [4], Factored RMAX [3] and Factored ϵ-greedy. The latter two methods are given the full dynamic Bayesian network (DBN) representation of the transition functions for both the *Get* and *Put*. We set the exploration threshold m to be 1 for both CTA-FRMAX and Factored RMAX. Since R-MAXQ does not converge with $m = 1$, we use $m = 5$ like other methods [4, 1].

As shown in Figure 2 and Figure 3, CTA-FRMAX outperforms all the other methods. In terms of number of episodes, CTA-FRMAX converges in about 50 episodes whereas R-MAXQ takes about 500 episodes to converge due to the delayed exploration—it is unable to explore *putdown* unless it completes the current navigation task. Also, R-MAXQ fails to converge to the optimal policy in the fickle taxi experiment, since it treats each subtask like a primitive action; it cannot immediately react to a change of destination until its current subtask completes. CTA-FRMAX also has better cumulative reward than Factored RMAX, despite that it only requires the relevant features for each task to be specified, while Factored RMAX requires the complete DBN structure of task transition dynamics.

Acknowledgments

This research is supported by a SMART Grant: Enhancing the Care for Elderly through Robotic Aides and Academic Research Grant: T1 251RES1211 from the Ministry of Education in Singapore.

REFERENCES

[1] F. Cao and S. Ray. Bayesian Hierarchical Reinforcement Learning. In *NIPS-12*, pages 73–81, 2012.

[2] T. G. Dietterich. The MAXQ Method for Hierarchical Reinforcement Learning. In *ICML-98*, pages 118–126, 1998.

[3] C. Guestrin, R. Patrascu, and D. Schuurmans. Algorithm-directed Exploration for Model-based Reinforcement Learning in Factored MDPs. In *ICML-02*, pages 235–242, 2002.

[4] N. K. Jong and P. Stone. Hierarchical Model-based Reinforcement Learning: R-MAX + MAXQ. In *ICML-08*, pages 432–439. ACM, 2008.

[5] R. Sutton, D. Precup, and S. Singh. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112:181–211, 1999.

[6] N. A. Vien and M. Toussaint. Hierarchical Monte-Carlo Planning. In *AAAI-15*, 2015.