An Efficient Approach to Model-Based Hierarchical Reinforcement Learning

Zhuoru Li,[†*] **Akshay Narayan,**[†] **Tze-Yun Leong**[†‡]
School of Computing, National University of Singapore[†]
School of Information Systems, Singapore Management University[‡]
lizhuoru@google.com, {anarayan, leongty}@comp.nus.edu.sg, leongty@smu.edu.sg

Abstract

We propose a model-based approach to hierarchical reinforcement learning that exploits shared knowledge and selective execution at different levels of abstraction, to efficiently solve large, complex problems. Our framework adopts a new transition dynamics learning algorithm that identifies the common action-feature combinations of the subtasks, and evaluates the subtask execution choices through simulation. The framework is sample efficient, and tolerates uncertain and incomplete problem characterization of the subtasks. We test the framework on common benchmark problems and complex simulated robotic environments. It compares favorably against the state-of-the-art algorithms, and scales well in very large problems.

Introduction

In hierarchical reinforcement learning (HRL), an agent solves large, complex problems by recursively decomposing the root problem into smaller tasks, and systematically solving the subtasks at different levels of abstraction. Existing HRL methods, such as options (Sutton, Precup, and Singh 1999), hierarchical abstract machines (HAMs) (Parr and Russell 1997), and MAXQ-based (Dietterich 1998) methods that include a pre-defined task hierarchy, assume different amount of prior domain knowledge in a task (or action) structure to speedup the search for good policies or solutions. The task-specific features or relevant state abstraction information must be given, through manual specification or automated learning, for all the methods to work effectively.

Existing HRL methods cannot efficiently solve many real-world problems involving multiple tasks, changing subgoals, and uncertain subtask specifications. For example, consider the duties of a household robot as shown in Figure 1. The robot performs different tasks at different locations: In the living room, switch off the television and light, and clean the table; in the yard, fetch mail and deliver it indoor, etc. Figure 2 shows a complex hierarchy of the robot's tasks.

We introduce a new model-based approach to HRL that addresses two major limitations of the MAXQ-based methods: First, the full set of tasks, including minor subtasks lower in the hierarchy, has to be clearly specified, which may be infeasible in real life. For example, to switch off the television, the robot first needs to navigate to the television location, which usually cannot be pre-specified. Second, multiple, similar subtasks have to be learned separately. For example, in Figure 3, there are multiple, independent navigation subtasks, one for each destination. The transition model learned for one subtask cannot be used on the others.

We propose a new framework called *context-sensitive reinforcement learning* (CSRL), that exploits common knowledge in the subtasks[1] to efficiently learn the transition dynamics. A main idea is to actively evaluate the different subtasks as execution choices based on simulation. CSRL is sample efficient and tolerates uncertain or incomplete specification of the task hierarchy; it can solve complex, multi-task problems with more than one million states. In practice, CSRL assumes the relevant features for each task are given, as in the MAXQ (Dietterich 1998) safe state abstraction: a state feature is considered to be irrelevant if it neither affects the transition of relevant features nor the reward received. CSRL can learn both the transition and reward functions; this paper focuses on learning transition functions.

Background: The mathematical model underlying all the HRL methods is the semi-Markov decision process (SMDP). Recall that a Markov decision process (MDP) is a tuple $\langle S, A, P, R\rangle$ where S is the set of states, A is the set of actions, P is the transition function that describes the probability of transiting to the next state, and $R(s,a)$ is the reward function for executing action a in s. In factored representation, a state s is specified as a list of state variables/features. For problems with infinite horizons, a discounted factor γ is introduced to bound the accumulated reward. An SMDP extends an MDP, such that each action may take multiple time steps to complete (Puterman 1994), i.e. $P(s', \tau|s,a)$ is the probability of transiting to a state s' in τ steps, after executing action a in s. The transition function can be computed by $P(s'|s,a) = \sum_j P(s', \tau = j|s,a)\gamma^j$.

A reinforcement learning (RL) (Kaelbling, Littman, and Moore 1996; Sutton and Barto 1998) agent plans and acts in an unknown environment modeled as an MDP to maximize the cumulative reward. We adopt the common R-MAX (Brafman and Tennenholtz 2003) directed exploration approach to balance exploration and exploitation.

[*] Currently affiliated with Google Korea, LLC

[1] Through contextual independence of state variables conditioned on the action executed in the underlying model, hence the name.

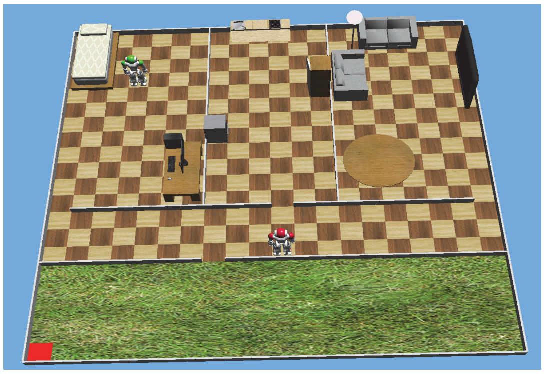

Figure 1: Household robot

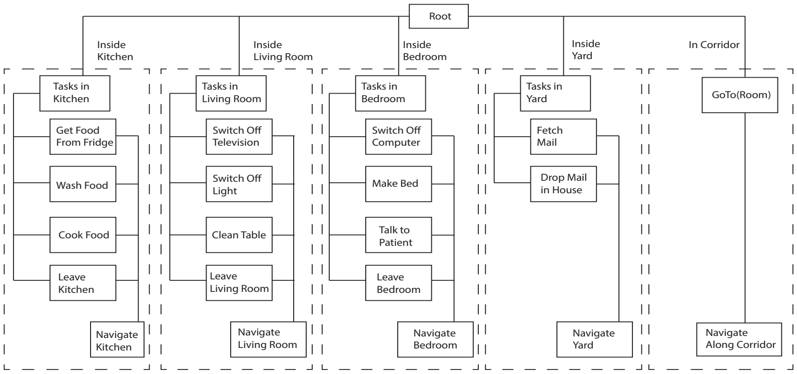

Figure 2: Task hierarchy for the household robot

Overview

In CSRL, we adopt the concepts from the MAXQ framework, but modify the definitions of *tasks* and *task hierarchy*. We further introduce the notion of *fragments* to allow better generalization and sharing of experiences.

Definition 1 (Task) *The tuple $\langle I_i, G_i, F_i, A_i, T_i, R_i\rangle$ represents task i, where I_i is the input set, which indicates when a task is applicable; G_i are the task goal states/terminating states; F_i is the set of relevant features, A_i is the set of applicable actions for task i, T_i and R_i are the transition and reward functions respectively, local to the task.*

A task is a unit of execution; it is a "smaller" MDP for accomplishing a subgoal of the problem. The definition is similar to an option (Sutton, Precup, and Singh 1999), where I_i and G_i corresponds to the option's initiation set and termination condition. The subset $\langle F_i, A_i, T_i, R_i\rangle$ of the task tuple forms a well-defined factored MDP.

Definition 2 (Fragment) *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.*

A fragment does not have goal states or local reward functions, and can be used to describe either multiple tasks that share the same transition function, but differ in goals or a task with incomplete specification (such as unknown goals). Fragments can be automatically generated by combining tasks with the same relevant features and actions. For example, in Figure 3, the MAXQ hierarchy includes multiple navigation subtasks, one for each possible destination. In contrast, in Figure 4, the CSRL hierarchy combines all the experiences into a single fragment to improve navigation, regardless of the destination. Fragments cannot be executed directly, it is used to facilitate efficient learning of task transition functions. However, the agent can directly execute the ancestor task of the fragment in the hierarchy.

The general CSRL hierarchy is shown in Figure 5. We define a *node* in the hierarchy to be either a task or a fragment. It is similar to a MAXQ hierarchy, but with two differences: (i) a task can be decomposed into fragments as well as smaller tasks; (ii) the primitive actions no longer appear as the leaf nodes, since they are defined inside each node. The set of relevant features for a node is always a subset of its parent node. This formulation requires no more information than the MAXQ hierarchy with state abstraction information, as introducing fragments allows the program to specify tasks without goal states.

Learning Task Transition Functions

The task learning mechanism in CSRL allows efficient learning of transition dynamics for every node in the task hierarchy. In CSRL, if two tasks share some common relevant features, then the experience in any task is used to improve the model in both tasks. Incorporating the fragments also allows experiences to be shared efficiently across tasks.

We use N to denote the set of non-root nodes in the hierarchy, and X is a subset of N. CSRL examines the features that are unique to a set of nodes, and partitions the set of state features into components.

Definition 3 (Unique function (Uni)) *The function Uni returns the unique features for a subset of nodes, X, under consideration. $Uni(\bigcap_{i\in X} F_i) = \bigcap_{i\in X} F_i - \bigcup_{j\in\overline{X}} F_j$.*

Definition 4 (Components of F) *For a subset of nodes X, $Uni(\bigcap_{i\in X} F_i)$ is called a component if it is non empty. The set of components form a partition over the set of features.*

The transition dynamics of any node in the hierarchy can be factorized into that of its components. We illustrate the transition function derivation using a taxi example: the agent controls a taxi, and its goal is to deliver the passenger to the destination. We formulate the problem using four state features: the location of the passenger p, the location of the taxi t, a binary variable in indicating whether passenger is in the taxi, and the location of the destination d. The *Get* task (task 1) uses the first three features while the *Put* task (task 2) uses the last three features. Both tasks share the navigation actions, and each task have a unique action to *pickup* or *putdown* the passenger. The four state features are divided into three components: $Uni(F_1)$, $Uni(F_2)$, and common features, $Uni(F_1 \cap F_2)$, i.e. (p), (d) and (in, t) respectively.

To compute the transition function, we consider the following cases.

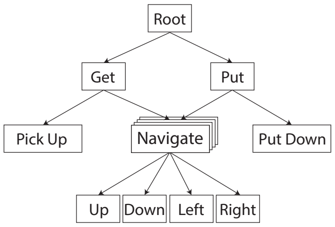

Figure 3: MAXQ hierarchy

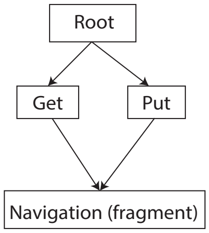

Figure 4: CSRL hierarchy

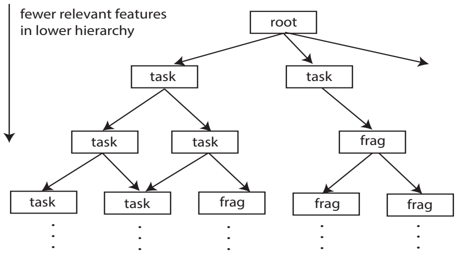

Figure 5: A general CSRL hierarchy

Action unique to one task: If the action executed is unique to task 1, i.e. the *pickup* action, the transition probability can be decomposed into two factors: features of task 1 (p, t, in), and features unique to task 2 (d). The agent first predicts the features in task 1. The value of the shared features $F'_1 \cap F'_2$ (in and t) only depends on the features of task 1, (F_1). The agent then predicts the unique features of task 2 (d), using F_2, i.e. $P(F'_1 \cup F'_2|F_1 \cup F_2, a) = P(F'_1|F_1, a)P(Uni(F'_2)|F_2, a_\varnothing)$, where $a_\varnothing$ is the default action if the action executed is not applicable to the task.

Action common to both tasks: If an action is common to both tasks (e.g., navigation actions), then the transition probability is constructed from three factors: features unique to task 1 (p), features unique to task 2 (d), and, features common to both tasks (in, t).

The agent first predicts $Uni(F'_1)$ based on the current values of F_1. The agent then predicts $Uni(F'_2)$ based on the current values of F_2. Finally the agent predicts the values of $F'_1 \cap F'_2$ based on their corresponding current values $F_1 \cap F_2$. Hence, if $a \in A_1 \cap A_2$,
$P(F'_1 \cup F'_2|F_1 \cup F_2, a) = P(Uni(F'_1)|F_1, a) \cdot P(Uni(F'_2)|F_2, a) \cdot P(F'_1 \cap F'_2|F_1 \cap F_2, a)$

The key observation is that factorization of the transition model depends on the current action being executed. We can generalize the previous example to n task problems. For large problems, we usually compute the task transition function instead of the global transition function, following Theorem 1.

Theorem 1 *Given an action a, let $X_a = \{i \,|\, i \in N \wedge a \in A_i\}$ be the set of nodes where action a is applicable. The transition function for a task i is given by*

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

The theorem states that the transition function for a task i is the product of the transition functions of its components. The theorem can be easily adapted to compute the global transition function by including every component. Algorithm 1 gives an overview of the CSRL algorithm. In the algorithm, m is the exploration threshold; $\mathcal{P}_{k,a}$ denotes the parent of component C_k with respect to action a (i.e., $\mathcal{P}_{k,a} = Par(C_k, a)$), and is initialized in line 6. The agent maintains an exploration count, $n(\mathcal{P}_{k,a}, a)$, for the parent feature and action pair. If the exploration count is smaller than the threshold, it transits to a fictitious component C_k^f with probability 1 (line 13), if not, probability values are updated. If a task is completed, a new task will be selected (line 18–21), which will be discussed later. It then constructs the model for task i and solve[2] it to obtain the next action to perform (line 23). Once the action is executed, the state of the system is updated (lines 25–28).

Algorithm 1 CSRL Algorithm

```
1:  Input: $m$, $F_i$ for all tasks $i$
2:  $s \leftarrow s_0, X_a \leftarrow \{i|a \in A_i\}$
3:  for all components $k$ do
4:      $X_{C_k} \leftarrow \{i|C_k \subseteq F_i\}$        //tasks using $C_k$
5:      for all actions $a$ do
6:          $\mathcal{P}_{k,a} \leftarrow \bigcap_{i\in X_{C_k}\cap X_a} F_i$
7:          $n(\mathcal{P}_{k,a}, a) \leftarrow 0$; $P(\cdot|\mathcal{P}_{k,a}, a) \leftarrow \varnothing$
8:      end for
9:  end for
10: while $s \notin$ terminal state do
11:     for all components $k$, action $a$ do
12:         if $n(\mathcal{P}_{k,a}, a) < m$ then
13:             $\hat{P}(C_k^f|\mathcal{P}_{k,a}, a) \leftarrow 1$
14:         else
15:             $\hat{P}\left(\cdot|\mathcal{P}_{k,a}, a\right) \leftarrow P\left(\cdot|\mathcal{P}_{k,a}, a\right)$
16:         end if
17:     end for
18:     if current task is completed or no task selected then
19:         ConstructSMDP($s$)
20:         Solve previous SMDP to get next task $i$.
21:     end if
22:     Construct model for task $i$ using Alg 1.
23:     Solve model for task $i$ to get a task policy $\pi_i$.
24:     Execute $(s, \pi_i(s)) \rightarrow s'$
25:     for all components $k$ do
26:         $n(\mathcal{P}_{k,a}, a) \leftarrow n(\mathcal{P}_{k,a}, a) + 1$
27:         Update $P(\cdot|\mathcal{P}_{k,a}, a)$ with $s$, $s'$
28:     end for
29:     $s \leftarrow s'$
30: end while
```

Alternatively the agent can compute the global transition function and line 18–21 is no longer necessary. Such variant can be considered as an improved version of Factored R-MAX: our method only requires the state abstraction information, which might be easier to specify than the full dynamic Bayesian network (DBN) structure as required by Factored R-MAX. Similar to the analysis of Factored R-MAX (Strehl

[2] We use value iteration in our implementation. Other methods such as prioritized sweeping can also be used to solve the model.

Algorithm 2 ConstructSMDP(current_state)

```
1:  $state\_queue$.enqueue(current_state)
2:  while $state\_queue$ not empty do
3:      $s$ = $state\_queue$.dequeue
4:      if $s$ is new then
5:          for all tasks $i$ do
6:              [$succ$, $reward$, $dist$] = SimulateTask($s$, $i$)
7:              if $\neg succ$ then
8:                  $R(s,i) \leftarrow R_{\max}$
9:                  $P(s|s,i) \leftarrow 1$
10:             else
11:                 $R(s,i) \leftarrow reward$
12:                 $P(\cdot|s,i) \leftarrow dist$
13:             end if
14:             for all states $s'$ in $dist$ do
15:                 state_queue.enqueue($s'$)
16:             end for
17:         end for
18:     end if
19: end while
```

Algorithm 3 SimulateTask(s, i)

```
1:  $distribution \leftarrow \varnothing$,$average\_reward \leftarrow 0$
2:  for $simulation\_num$: 1 to NUM_SIM do
3:      $reward \leftarrow 0$; $steps \leftarrow 0$
4:      while True do
5:          $a \leftarrow \pi_i(s)$; $reward \leftarrow reward + R_i(s,a)$
6:          $s \leftarrow$ SampleRootTransition ($s$, $a$)
7:          $R(s,i) \leftarrow R_{\max}$
8:          if $steps >$ NUM_STEPS or $s$ is fictitious then
9:              return [False, Null, $\varnothing$]
10:         end if
11:         if $s$ is goal state then
12:             Update $average\_reward$ with $reward$
13:             Update $distribution$ with $s$ as terminal state
14:             Update $distribution$ with duration to complete
15:             break
16:         end if
17:     end while
18: end for
19: return [True, $average\_reward$, $distribution$]
```

2007), we can derive the sample complexity of using a global transition function, ignoring log terms:

$$\mathcal{O}\left(\frac{\mathscr{C}^2 V_{\max}^3 \sum_{k,a} |\mathcal{D}(Par(C_k,a))||\mathcal{D}(C_k)|}{\epsilon^3(1-\gamma)^3}\right)$$

where $|\mathcal{D}(C_k)|$ is the number of possible values for component C_k, $V_{\max} = R_{\max}/(1-\gamma)$ and $\mathscr{C}$ is the number of components.

State space size: The sample complexity does not depend on the size of the state space; the sample complexity is linear in the number of parameters in the factorization.

Optimality: The sample complexity analysis implies that if we solve the learned root transition model, the value function is ϵ-optimal to the actual value function with high probability, except for a small number of steps.

Managing Hierarchical Execution

At the beginning of each episode, or after completing the current task, the agent must select a new task to execute (line 18–21 in Alg 1). We model task selection as an SMDP, as shown in Alg 2. The main difference from other approaches is on how the SMDP parameters are estimated. Since the transition function is computed for every node in the hierarchy, the parameters can be more efficiently estimated by simulating the effect of executing the task policy on the task's parent node's transition function (Alg 3).

In Alg 2, the agent maintains a queue of states to be simulated and proceeds as follows. When the queue is not empty, a state s is extracted from the queue (line 3). If the state is new, the parameters $R(s,i)$ and $P(\cdot|s,i)$ are constructed through simulation (line 6). If the simulation for executing task i in state s fails, $R(s,i)$ is set to maximum reward $R_{\max}$ and $P(s|s,i)$ to 1 (line 8–9). Hence this state is now a fictitious state and it will be chosen to be executed. If the simulation succeeds, the agent receives the average reward and the distribution of terminating states, and constructs the SMDP transition function and reward function using these values (line 11–12). Then all the terminating states from simulation are inserted into the state queue and the process repeats. In the Alg 3, inside the while loop, the agent obtains an action from the task policy (line 5) and samples a next state from the root transition function (line 7). The procedure returns false if the simulation fails to complete or reaches a fictitious state in the task (line 9).

Given the policies of the child tasks, CSRL can simulate the results of executing the task on the root node. Two issues should be noted. First, the simulation is computationally fast. The agent is only following a task policy without additional computations. We report the running time in the experiments section. Second, if a task simulation fails, the agent receives maximum reward and will explore the task, leading to a better task policy. The simulations for a task will not fail forever.

The procedure `ConstructSMDP` can be recursively called at any level to construct the parameters. Given a task policy for any node, we can construct the task selection SMDP at the parent node. Solving the task selection SMDP results in a policy for the parent node, which can be used for simulation at an even higher level.

Multiple CSRLs can also be applied in one problem. Multi-CSRL is a simple extension where a CSRL instance is treated like a single action by its parent. Multi-CSRL is suitable for solving problems that can be decomposed into multiple independent sub-problems, each of which can be modeled as a single CSRL instance. In the household robot example, each room is modeled as a CSRL independent of the others.

Related Work

Recent developments in HRL are mainly based on the options (Sutton, Precup, and Singh 1999) and the MAXQ (Dietterich 1998) frameworks. Some efforts have focused on learning options or transferring them across multiple MDPs. Mann et al. (2014) introduce regularization to options that favor longer duration skills. Silve and Ciosek (2012) propose recursive composition of option models into other option models.

However, this work focuses on known MDP rather than reinforcement learning problems. Levy and Shimkin (2011) use actor-critic framework that allows concurrent inter- and intra-options learning. Brunskill and Li (2014) study the sample complexity of SMDP-RMAX algorithm, and use PAC analysis to prune options for knowledge transfer. Konidaris and Barto (2009) propose skill chaining in continuous RL domains, that segments complex policies into skills that can be executed sequentially. Bai et al. (2016) show that an MDP with state abstraction can be converted into a POMDP and solved efficiently using hierarchical Monte Carlo Tree Search; this work also does not focus on reinforcement learning.

MAXQ based methods share the same task hierarchy, but adopt different solution methods. R-MAXQ is a model-based approach that combines the MAXQ task hierarchy and R-MAX exploration (Jong and Stone 2008), and uses prioritized sweeping to solve the learned model. The more recent Bayesian MAXQ incorporates Bayesian prior to the MAXQ algorithm (Cao and Ray 2012), but the assumption on the given priors is not feasible in many domains. Vien et. al (2014; 2014; 2015) introduce H-UCT that extends MAXQ with hierarchical Monte-Carlo simulation. However, Bayesian approaches can be computationally intensive.

Experiments

We test the empirical performance of CSRL on a set of benchmark experiments, formulated as a robot HRL agent solving different tasks. We design the experiments with increasing state sizes to demonstrate that the proposed methods are scalable to handle complex problems. The first experiment does not require task selection, while the last large-scale, household robot experiment requires multiple levels of reasoning. Moreover, the last experiment cannot be solved using existing methods due to incomplete problem specification at the lower level: the terminating states for indoor navigation is not known beforehand. We do not compare our work with the recent Bayesian MAXQ or H-UCT as they are computationally inefficient in solving the standard benchmarks (Vien and Toussaint 2015). In all experiments, an episode terminates if it does not complete in 1000 steps.

Robot Pickup and Place: The first experiment involves a robot moving objects to designated locations. This is a variant of the HRL benchmark Taxi problem (Dietterich 1998). We use the 10×10 grid world from Diuk *et al.* (Diuk, Cohen, and Littman 2008). The problem has 7200 states. The goal is to pick up an object and place it at the a designated location. Both the initial and the designated locations of the object are on the landmarks. The problem is decomposed into *Get*, where the robot picks up the object; and *Put*, where the robot puts the object at the designated location. There are six actions: four navigation actions, *pickup* to pick up the item, and *putdown* to put down the item. Each navigation step costs -1 and a successful *putdown* leads to +20; executing *pickup* and *putdown* at the wrong state leads to -10.

In this experiment, the agent must first complete *Get* before starting *Put*, thus task selection is not needed.

We compare our method against R-MAXQ, Factored R-MAX and Factored ϵ-greedy, which replaces the directed exploration in Factored R-MAX with ϵ-greedy exploration. Factored R-MAX and Factored ϵ-greedy are given the task DBN structures for both *Get* and *Put*. CSRL and R-MAXQ are given the relevant feature information. We set the exploration threshold, $m = 1$ for all methods. Since R-MAXQ cannot converge with $m = 1$, we set $m = 5$ like other existing works (Jong and Stone 2008; Cao and Ray 2012).

The results, averaged over 100 runs, are shown in Figure 6. CSRL converges to the optimal policy in fewer than 100 episodes, while R-MAXQ does not converge even after 1000 episodes. R-MAXQ cannot explore until the subtask completes. Hence, it requires larger number of episodes to complete the exploration. CSRL fares better than Factored R-MAX, despite the fact that it does not know the DBN structure. Factored ϵ-greedy converges to a near-optimal policy. Due to ϵ-greedy exploration it never stops exploring, and thus fails to reach the optimal policy.

Pickup and Place—Two Objects: We define an extension of the previous problem in which there are two objects in the environment. The agent now has two *pickup* and two *putdown* actions, and the robot can pick up both objects at the same time. This experiment contains 518400 states. We do not compare with R-MAXQ as it needs at least 1472 episodes to complete exploration (92 non-landmark locations, 8 designated locations, 2 objects).

We compare CSRL with Factored R-MAX Random and Factored ϵ-greedy Random. These two methods use random task selection. We also compare with SMDP-RMAX (Brunskill and Li 2014) for task selection, but use Theorem 1 for task learning. The results of an average of 100 runs are shown in Figure 7. CSRL has the best results, both in terms of average and accumulated rewards. CSRL and SMDP-RMAX have almost the same performance for the first 50 episodes, as they use the same task learning approach. Once CSRL has learned the accurate task dynamics for all the tasks, it converges to a better policy than SMDP-RMAX, as it can utilize the root transition function to construct the parameters for the SMDP at the top level. For the 500 episodes tested, SMDP-RMAX performs similarly to random task selection, showing that it requires many more samples to converge. We have also tested CSRL with up to eight objects in various settings, and found that it performs well in all the experiments.

Household Robot Experiment: The final experiment is the household robot problem presented in the Introduction, modeled using Webots (Michel 2004) simulator. Recall that this problem cannot be modeled using existing HRL methods, as the destination of indoor navigation is not known beforehand.

The state variables or features are the robot's location and orientation, as well as 10 binary variables that indicate whether a task is complete. The size of the state space is 1638400 (400 cells, 4 orientation, 10 binary variables). There are 17 actions: move forward, turn left/right, one unique action for each of the 10 tasks (such as turning off the television), and one action to open the door for each room. The reward for navigation actions and opening doors is -1. The reward for the tasks' unique actions is 40 if it completes the task, and -5 for attempting actions at wrong locations. The navigation actions have stochastic effects, while the others are deterministic.

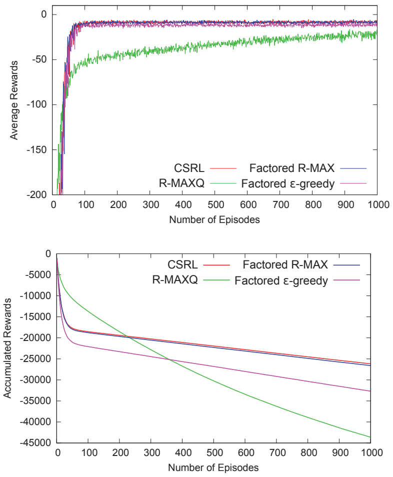

Figure 6: Robot pickup and place

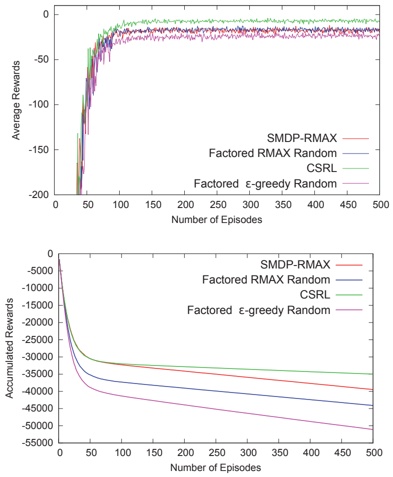

Figure 7: Pickup and place: 2 objects

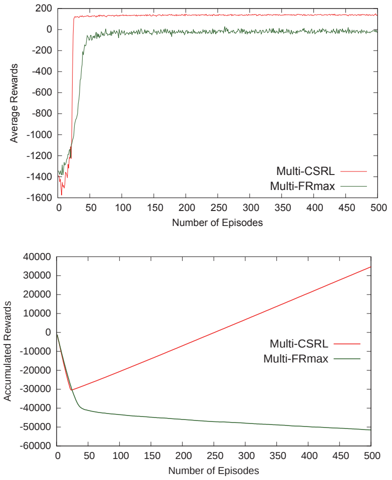

Figure 8: Household assistive robot

We model the problem as a Multi-CSRL, which treats each CSRL instance as a single action. Each dotted rectangle in Figure 2 represents a CSRL instance. The root level has eight actions: each of the four rooms has its own CSRL and a *GoTo(Room)* action in the corridor. The agent chooses actions by constructing a root level SMDP, which is similar to the task selection SMDP in a CSRL. We run the experiment using Multi-CSRL and multiple Factored R-MAX (Multi-FRMAX) with the exploration threshold $m = 1$ for both methods. Multi-FRMAX is extended in a similar way but it follows a fixed order of execution—living room, kitchen, bedroom and yard; if the starting location is in a room, it completes all tasks in the room, enters the corridor, and completes the remaining tasks according the list.

The results are shown in Figure 8. The most important observation is a huge improvement on average performance for Multi-CSRL at around the 25th episode, the value of the policy increases from about -1200 to more than 100. This is because Multi-CSRL features very efficient multi-layer directed exploration: task learning first indicates an unknown component-action pair by making it transit to a fictitious state, reflected in the value function for all the tasks that includes this component-action pair. The hierarchical execution of the room CSRL treats these tasks as another layer of fictitious states, and directs the agent to explore these tasks. Finally, the root level notices the room CSRL having fictitious states, and directs the agent to visit the room. Once the exploration is complete, a near optimal policy is immediately reached. Instead, Multi-FRMAX converges to a sub-optimal policy, indicating that SMDP task selection leads to much better performance than fixed order task selection. Multi-CSRL also has better accumulated reward, as sharing the navigation fragments minimizes the amount of exploration required. Unlike Multi-CSRL, which has a sudden improvement in average reward, Multi-FRMAX's improvement in average reward is more smooth. This shows that without a multi-layer directed exploration mechanism as in Multi-CSRL, more episodes are required to complete all the exploration steps.

In terms of computational time, both methods have similar performance. The running time is the average of 10 independent runs, on a Xeon E5-2643 v2 3.50GHz using a single thread. Multi-CSRL completes 100 episodes in 610.8 seconds, while Multi-FRMAX completes in 616.7 seconds. When a task is selected, both Multi-CSRL and Multi-FRMAX have similar per-step computation cost as both use value iteration to solve a task. However, Multi-CSRL learns a better policy which involves smaller number of steps per episode, and this also shows that simulation is computationally efficient and does not result in a longer running time.

Discussion and Conclusion

We have introduced CSRL, which includes a task learning mechanism that learns both the task and the global transition dynamics, and a hierarchical execution mechanism handling task selection by formulating and solving the underlying SMDP. We have empirically shown that, by simulation on learning the global transition functions, CSRL performs much better than other state-of-the-art algorithms, and is scalable to problems with over one million states. CSRL is also able to handle various problems that cannot be solved by many existing HRL methods. For ease of exposition and as in most MAXQ based HRL methods, we have assumed safe state abstraction with manual specification of the relevant features. This assumption can be relaxed by extending the framework with automated feature learning and feature selection mechanisms. Future work could also include knowledge transfer to target problems that share similar hierarchies. Each CSRL instance in the Multi-CSRL setting can be abstracted as a rule and transferred across similar domains.

Acknowledgments

This research was partially 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

Bai, A.; Srivastava, S.; and Russell, S. J. 2016. Markovian state and action abstractions for MDPs via hierarchical MCTS. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, 3029–3039.

Brafman, R. I., and Tennenholtz, M. 2003. R-MAX—A General Polynomial Time Algorithm for Near-optimal Reinforcement Learning. *The Journal of Machine Learning Research* 3:213–231.

Brunskill, E., and Li, L. 2014. PAC-inspired Option Discovery in Lifelong Reinforcement Learning. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 316–324.

Cao, F., and Ray, S. 2012. Bayesian Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems (NIPS-12)*, 73–81.

Dietterich, T. G. 1998. The MAXQ Method for Hierarchical Reinforcement Learning. In *Proceedings of the 15th International Conference on Machine Learning (ICML-98)*, 118–126.

Diuk, C.; Cohen, A.; and Littman, M. L. 2008. An Object-oriented Representation for Efficient Reinforcement Learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML-08)*, 240–247. ACM.

Jong, N. K., and Stone, P. 2008. Hierarchical Model-based Reinforcement Learning: R-MAX + MAXQ. In *Proceedings of the 25th International Conference on Machine Learning (ICML-08)*, 432–439. ACM.

Kaelbling, L. P.; Littman, M. L.; and Moore, A. W. 1996. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research* 237–285.

Konidaris, G., and Barto, A. G. 2009. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems (NIPS-09)*, 1015–1023.

Levy, K. Y., and Shimkin, N. 2011. Unified Inter and Intra Options Learning Using Policy Gradient Methods. In *Proceedings of 9th European Workshop on Recent Advances in Reinforcement Learning (EWRL-11)*, volume 7188 of *Lecture Notes in Computer Science*, 153–164. Springer.

Mann, T.; Mankowitz, D.; and Mannor, S. 2014. Time-regularized interrupting options (TRIO). In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 1350–1358.

Michel, O. 2004. Webots: Professional Mobile Robot Simulation. *Journal of Advanced Robotics Systems* 1(1):39–42.

Parr, R., and Russell, S. 1997. Reinforcement Learning with Hierarchies of Machines. In *Advances in Neural Information Processing Systems (NIPS-97)*, 1043–1049. The MIT Press.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1st edition.

Silver, D., and Ciosek, K. 2012. Compositional planning using optimal option models. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*.

Strehl, A. L. 2007. *Probably Approximately Correct (PAC) Exploration in Reinforcement Learning*. Ph.D. Dissertation, Rutgers University-Graduate School-New Brunswick.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. The MIT Press.

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

Vien, N. A., and Toussaint, M. 2015. Hierarchical Monte-Carlo Planning. In *Proceedings of The 29th AAAI Conference on Artificial Intelligence (AAAI-15)*.

Vien, N. A.; Ngo, H.; Lee, S.; and Chung, T. 2014. Approximate Planning for Bayesian Hierarchical Reinforcement Learning. *Applied Intelligence* 41(3):808–819.

Vien, N. A.; Ngo, H.; and Wolfgang, E. 2014. Monte carlo bayesian hierarchical reinforcement learning. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, 1551–1552. International Foundation for Autonomous Agents and Multiagent Systems.