Partial Order Hierarchical Reinforcement Learning

Bernhard Hengst

Neville Roach Research Laboratory, NICTA, Sydney Australia
School of Computer Science, University of NSW, Sydney, Australia

Abstract. In this paper the notion of a partial-order plan is extended to task-hierarchies. We introduce the concept of a partial-order task-hierarchy that decomposes a problem using multi-tasking actions. We go further and show how a problem can be automatically decomposed into a partial-order task-hierarchy, and solved using hierarchical reinforcement learning. The problem structure determines the reduction in memory requirements and learning time.

1 Introduction

Russel and Norvig [1] note that "Fortunately, the real world is a largely benign place where subgoals tend to be nearly independent". Polya [2] exploits this real-world property, suggesting that a good problem solving heuristics is to ask "Can you decompose the problem and recombine its elements in some new manner?" The decomposition of a problem is usually left to a human designer and a challenge in machine learning is to automate this divide-and-conquer technique.

On a related theme it has often been stated that the complexity we encounter in natural environments takes the form of hierarchy and that hierarchy is one of the central structural schemes that the architecture of complexity uses [3, 4]. Hierarchical reinforcement learning (HRL) uses hierarchy to represent and solve problems more efficiently (see [5] for a survey). One HRL approach, MAXQ [6], explicitly uses a task-hierarchy.

Several researchers have looked at automating the decomposition of a reinforcement learning problem into subtasks and recombining their policies to solve the original problem [7–11]. HEXQ [11] is a decomposition algorithm that automatically builds a MAXQ-like task-hierarchy by uncovering structure in problems from state variables. It decomposes a problem by ordering the state variables by their frequency-of-change, constructing a task-hierarchy with one level per variable.

This paper introduces partial-order task-hierarchies using multi-tasking actions that execute several independent subtasks in arbitrary order. The partial-order generalisation of task-hierarchies can represent hierarchical reinforcement problems more succinctly and speed up learning time. We address shortcomings in HEXQ by examining several state partitions of the problem at the same time and by building a more compact partial-order task-hierarchy representation.

We assume the reader is familiar with reinforcement learning [12] and hierarchical reinforcement learning [5]. In the rest of this paper we introduce partial-order task-hierarchies. We describe how a two-level partial-order task-hierarchy is constructed using a simple one-room problem to motivate the discussion. Finally, we show a diverse set of results, using several room mazes, and demonstrate savings in learning time and memory requirements for a larger "breakfast" task.

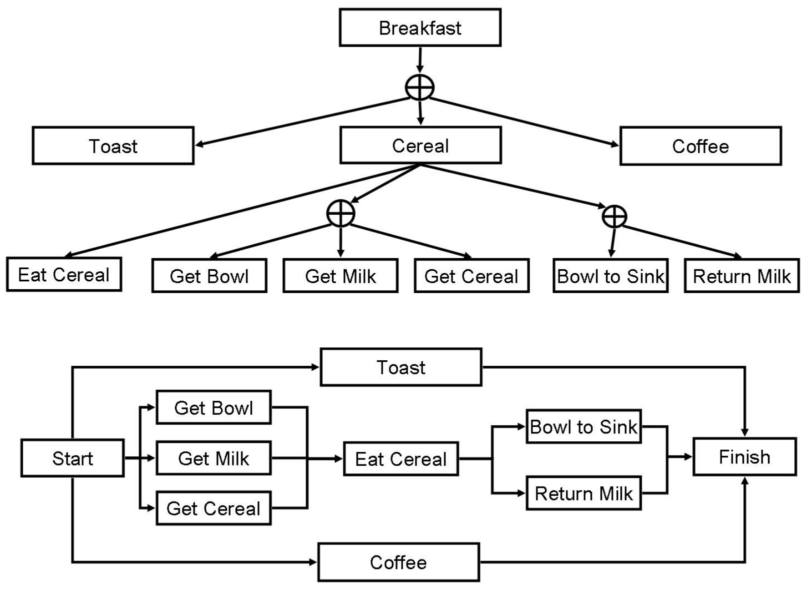

Fig. 1. Top: A partial-order task-hierarchy for a breakfast task. Bottom: The optimal partial-order plan.

2 Partial-Order Task-Hierarchies

In a task-hierarchy each action invokes one subtask. In a *partial-order* task-hierarchy we allow actions to invoke several independent subtasks. We call these actions *multi-tasking* actions, because their objective is to complete multiple subtasks before terminating. Multi-tasking actions can be interpreted as committing simultaneously to multiple subgoals, but they are only able to pursue one subgoal at each time-step.

An algorithm that can represent a problem using multiple actions without specifying their order of execution is a partial-order planner [1]. Multi-tasking actions are non-deterministic in this way and can be used to extend the notion of a task-hierarchy to a partial-order task-hierarchy. We indicate multi-tasking actions in a partial-order task-hierarchy graph by the symbol $\bigoplus$ and coalesce edges from the parent task to multiple child subtasks, signifying that all the child subtasks need to complete and terminate before returning control to the parent.

Figure 1 (top) shows an example of a partial-order task-hierarchy for an agent that plans to consume cereal, toast and coffee in some arbitrary order for breakfast. The cereal subtask is further divided into (1) a simple action to eat the cereal, (2) a multi-tasking abstract action to co-locate a bowl from the cupboard, milk from the fridge and cereal from the pantry, and (3) another multi-tasking action to return the milk to the fridge and to take the bowl to the sink.

A partial-order task-hierarchy has the expressive power to represent multiple partial-order plans. For example, in the breakfast task, the order of the three actions available to the cereal subtask, and indeed whether they are needed at all, will be learnt by the hierarchical reinforcement learner. One of the possible partial-order plans, the optimal one, is shown in the bottom of Figure 1.

2.1 The One-Room Problem

We use reinforcement learning (RL) to represent these type of problems. RL is formalised as a Markov decision problem MDP $< S, A, T, R >$ where S is a finite set of states, A a finite set of actions, $T : S \times A \times S \rightarrow [0, 1]$ a stochastic transition function and $R : S \times A \times \mathbb{R} \rightarrow [0, 1]$ a stochastic reward function. We assume we are provided with states represented as a tuple of variables $(s^1, s^2, ..., s^i, ..., s^n)$. A *policy* is a function that specifies which action to take in any state. We use the action-value function $Q(s, a)$ as the expected undiscounted sum of rewards to termination, starting in state s, taking action a and following a specified policy thereafter. The reader is referred to [12] for further background on RL. In hierarchical reinforcement learning (HRL) parent task state actions can invoke subtasks [5]. Since a subtask can execute for multiple time-steps the parent problem becomes a semi-MDP.

We will use the simple one-room problem in Figure 2 as a running example. In this problem an agent starts at random in a 10 x 10 grid-world room. Its aim is to leave the room by the shortest path. The state is the position of the agent perceived as coordinates (x, y). Although x and y take integer labels they are not assumed to have any numeric meaning. In each state the agent can choose from four stochastic actions - move one step to the North, East, South or West. Actions move the agent in the intended direction 80% of the time, but 20% of the time the agent stays where it is. The reward at each step is -1.

Solving this problem in a straightforward manner using reinforcement learning requires a table of Q values of size $|S||A| = 100 * 4 = 400$. However, the agent could learn to move in the x and y directions independently. The only dependence is at $(2, 9)$ where a move to the North reaches the goal. This results in the decomposition shown by the partial-order task-hierarchy in Figure 2 (right). The root subtask consists of 1 abstract state representing the room. The root multi-tasking action invokes two subtasks, one for navigating North-South, the other East-West. We terminate the multi-tasking action when the North-South

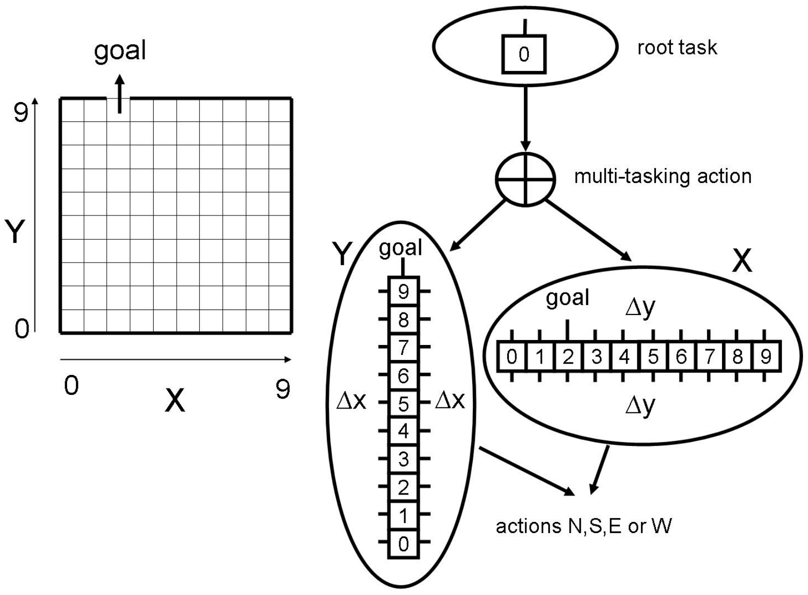

Fig. 2. The one-room problem and its decomposition using a partial-order task-hierarchy with one multi-tasking action.

subtask state is 9 and the East-West subtask state is 2 followed by the action to move North. The number of Q values required to represent this partial-order task-hierarchy by a hierarchal reinforcement learner is reduced from 400 to only 81 values as will be shown later.

3 Constructing Partial-Order Task-Hierarchies

We will show how our hierarchical reinforcement learner constructs and solves partial-order task-hierarchies taking a closely related approach to HEXQ [11]. HEXQ does not "fail" on any of the tasks in this paper but it can be inefficient because each variable in the state description is forced to add another level to the task-hierarchy. Multi-tasking actions overcome this issue.

Multi-tasking actions implement a similar function to the AND decomposition in StateCharts [13] that capture the property that the system state must be simultaneously in each subtask being multi-tasked. The system state is the orthogonal product of the state variables describing the subtasks. When an action in one of the subtasks does not affect other subtasks it represents a kind of independence which we exploit to decompose MDPs extending the idea of StateChart AND decomposition to stochastic actions.

To discover partial-order task-hierarchies we process all variables in the state tuple concurrently to construct the first level in the hierarchy. For each variable we find the regions and region exits as in HEXQ [11]. In this paper we restrict ourselves to two level hierarchies, but the construction can be extended to multiple levels.

The root task state is also described by a tuple of variables, but now they identify the regions the system state is in at the level below. The root task state is therefore more abstract and factors out detail represented in the child subtask regions. If the problem does not have any structure, regions will be singleton states and there will be no computational gain in this approach. Once regions and region exits have been identified for all state variables we can set about constructing a set of multi-tasking actions for the root task.

The original task is solved by solving the top level semi-MDP in the abstract state-space using the multi-tasking actions. To solve the semi-MDP the HRL has to commit to a linearization of the multiple tasks. We arbitrarily use the state-tuple variable order, but one can imagine if this approach is used as an approximation, the commitment to the order of processing may be governed by a higher level context. We are afforded the same advantages in the flexibility of execution as for partial-order plans.

3.1 Regions and Exits

We now look at the construction in more detail. A *model* (the transition and reward function) is discovered for each state variable s^i using a random exploration policy as for HEXQ [11]. For each variable we store the transition function as a series of transitions from one state to the next for each action. We also tally the frequency as we experience the transition multiple times and accumulate the rewards to allow us to estimate the transition distribution and expected reward functions.

As in HEXQ an *exit* is a state-action pair used to signify that the transition or reward function cannot be well defined because, for example, there is state aliasing in the projected state space. We declare a transition an exit whenever another variable $s^l, i \neq l$ changes value or the problem terminates. For consistency, problem termination is represented as a boolean goal variable changing value from false to true. Importantly, and in contrast to HEXQ, we also store the set of variables s^l that change value for each exit.

Our hope is that the variables s^i show some independence and that we can discover independent models over at least part of their state-action space. The state-space for each variable is partitioned into regions (i.e. independent model subspaces) using the same criteria as for HEXQ . A *region* is a block of the partitioned state-space ensuring that all exit-states are reachable from all entry states.

As an example, the ovals marked X and Y in Figure 2 show the two projected state-spaces for the variables x and y. In this problem each projected state-space is just one region. Exits are indicated by short lines emanating from each of the 10 states. For the Y region the exits may change the x values and state $y = 9$ may additionally exit to the goal. Similarly, the X region has exits from all states that may change the y variable values.

3.2 Multi-tasking Actions

Multi-tasking actions are automatically generated by examining the exit information from all the regions. They are created by considering all possible combinations of exits for those regions that comprise each abstract root-task state. Multi-tasking actions are represented as a tuple $(s^1, ..., s^n, a)$ indicating the exit state s^i each subtask should reach before executing the exit action a. The procedure for creating multi-tasking actions is as follows:

- for each abstract state in the root task we identify the unique region associated with each variable s^i. This is straightforward because of the root task abstract state is a tuple of variables indexing these regions (see Section 3)
- we generate all possible combinations of exits, one from each identified region. Region exits are available from the procedure described in Subsection 3.1
- a multi-tasking action is created whenever all the exits in a combination:
  1. have the same exit-action, and
  2. change the same variables in the set of changed variables - the s^l in Subsection 3.1

To justify the first criteria above, we note that we require one action to terminate each multi-tasked subtask to return control to the parent task. Only one action can be performed at a time by the agent and therefore the exit-action must be common to all exits.

The system is simultaneously in all subtasks invoked by a multi-tasking action. When the multi-tasking action terminates it simultaneously terminates all subtasks. A valid multi-tasking action must therefore potentially change the same set of variables, justifying the second criteria. Together these criteria eliminate a large number of combinations.

As for HEXQ, we need to define and solve the MDP associate with each region in the multi-tasking action with the goal of reaching the exit-state and executing the exit-action.

Let's illustrate the construction using the one-room example. The two regions X and Y have 21 and 20 exits respectively as shown in Figure 2. Exit combinations include $\{(x = 4, a = North), (y = 9, a = North)\}$. This combination does not qualify for producing a multi-tasking action because the exits fail to change the same variable s^l. The goal variable is changed by $(y = 9, a = North)$, but not by $(x = 4, a = North)$.

In this problem it turns out that the only exit combination that meets the two criteria for multi-tasking actions is $\{(x = 2, a = North), (y = 9, a = North)\}$ producing only one multi-tasking action $(x = 2, y = 9, a = North)$. Note that in this case they both have exit-action $a = North$ and they both change the same variables, i.e. x, y and the goal.

The X region MDP takes the agent to $x = 2$ and the Y region MDP takes the agent to $y = 9$. The multi-tasking action is completed by executing action $North$. The number of Q values for the root task is $|S||A| = 1 * 1 = 1$ and $|S||A| = 10 * 4 = 40$ for each of the child subtasks, making a total of only 81

Table 1. Summary statistics for the partial-order task-hierarchical decomposition for the various mazes. $|S|$ is the number of abstract states discovered, $\overline{A/S}$ the average number of multi-tasking actions per abstract state generated, $\sum|S||A|$ shows the memory requirements to represent the abstract problem, $\sum|S||A|\%$ the size of the abstract subtask as a % of the original problem size and Tot% is the decomposed problem size as a % of the original problem.

Maze	$\|S\|$	$\overline{A/S}$	$\sum\|S\|\|A\|$	$\sum\|S\|\|A\|\%$	Tot%
(a)	4	10.3	41	10.25	110.3
(b)	9	9.0	81	20.25	92.3
(c)	10	3.7	37	9.250	49.3
(d)	90	3.4	305	76.25	98.3
(e)	4	4.0	16	4.000	110.3
(f)	4	6.8	27	6.750	158.8
Fig 2	1	1.0	1	0.250	20.3

vales required to store the one-room task. In comparison HEXQ requires a total of 610 Q values.

The total reward for a multi-tasking action is the cumulative reward from associated subtasks because subtasks are independent and executed serially. As in HEXQ, policies for problems with deterministic actions are optimal and hierarchically optimal for stochastic actions. Hierarchical optimality stems from having single exit subtasks ensuring that the policies are the best possible given the constraints of the hierarchy [11].

4 Results

We now use several mazes and a larger breakfast task to demonstrate the versatility of automatic partial-order HRL.

4.1 Mazes

The six mazes in Figure 3 are solved using partial-order HRL. In each case we assume a similar problem formulation to the one-room problem. Mazes (a) to (d) represent the state-space with a coordinate like attribute pair (x, y). Maze (e) is similar to the others except that X now takes on 25 different values to represent the position in each of the rooms and Y has 4 values to identify the room. In maze (f) we mix the two representations from mazes (a) and (e). The left half of the maze is described by coordinates and the right half, by room and position-in-room variables.

We are careful to ensure that state values are not aliased as this would result in a partially observable problem. In each case the order of the variables defining the state vector is arbitrary and any order will give the same results.

The maze in Figure 3 (a) is a four-room problem. This maze generates two regions for each of the variables X and Y as shown in Figure 4. Interestingly the root task that is generated has four abstract states representing the four

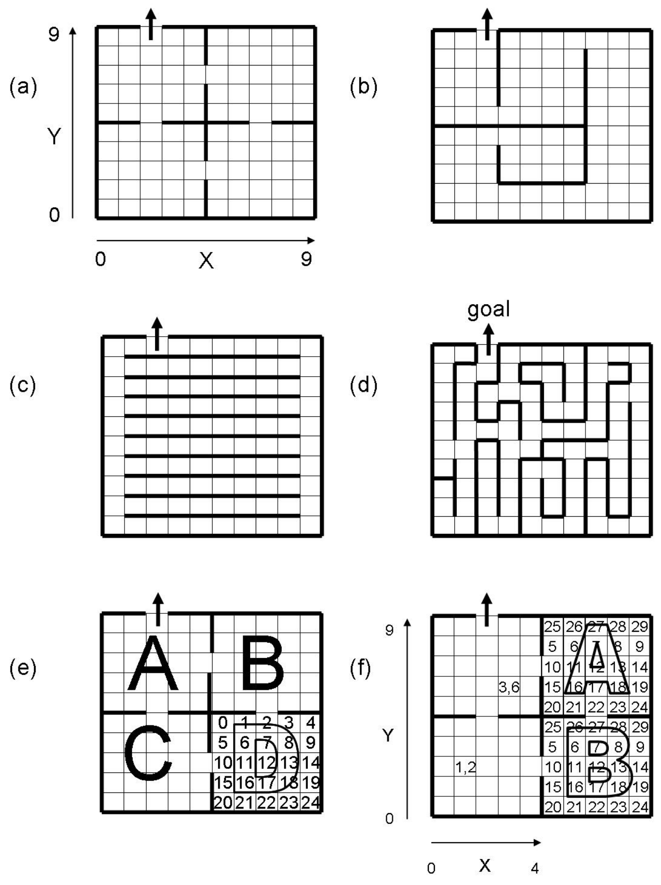

Fig. 3. Mazes used for Partial-Order HRL.

rooms in the domain. This demonstrates an ability of the algorithm to represent the problem at an abstract level to reflect the structure. There are 11 multi-tasking actions generated for the top-left-room state and 10 for the other three abstract room states. Most of these multi-tasking actions bump the agent into the walls and do not lead the agent out of the rooms. These effects are learnt as the top-level semi-MDP is solved.

Maze (b) decomposes into three regions per variable and hence 9 abstract root level states. Agents starting in the center region of the maze need to follow a spiral like path to reach the goal and learn to cut corners in the lower part of the maze to minimise the distance to the goal.

Maze (c) has one region for the X variable, 10 regions for the Y variable and an average of 3.7 multi-tasking actions per abstract state.

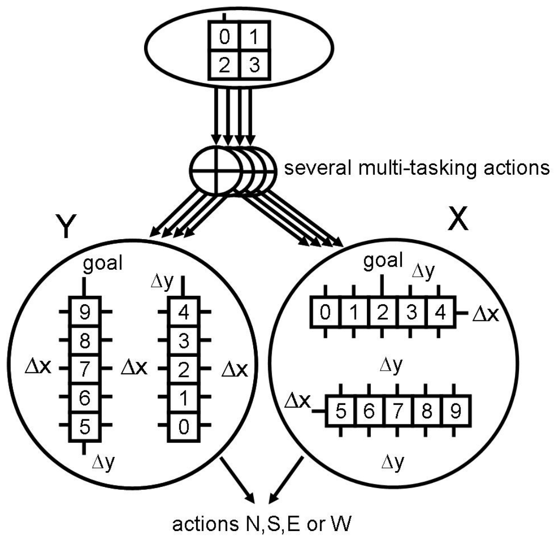

Fig. 4. The automatic decomposition of the four-room problem of maze (a) in Figure 3

Maze (d) illustrates a more complex situation where it might appear at first that abstraction is not possible. The algorithm found the two-state region, $[y = 1, y = 2]$, and reduced the number of multi-tasking actions for most abstract states due to the constraints in the maze, resulting in a slight reduction in overall memory requirements and the abstract problem reduced to 76% of the original.

Mazes (e) and (f) each use different representations for the base-level states and each still automatically decomposes to an abstract problem with four states representing the four rooms. While the number of abstract states is the same and they refer to the same rooms, the subtasks invoked by their multi-tasking actions vary. In contrast to maze (a), maze (e) in Figure 3 generates 1 region for the X variable and 4 regions for the Y variable. At the abstract level the Figure 3 (e) agent would explore a possible bottom-left room exit to the west, whereas the maze (a) agent would not. The latter already represents the impossibility of a West exit at $x = 0$, while the former knows it may be able to exit West at $x = 10$, but still needs to learn in which rooms.

Table 1 summarises the abstraction results for the room mazes. Memory requirements and problem sizes are measured in terms of the number of table entries required to store the value function. In each case we can find an abstract representation of the problem that is smaller than the original. Some of the mazes require more total storage after decomposition when subtasks are included, but we must remember that in practice problems will be larger with the cost of subtask storage amortised over many more contexts. For example, in maze (e) with four rooms the total partial-order task-hierarchy memory requirements exceed that of the original problem. The total memory requirements for n rooms of size 5 by 5 is $100n$ for the original problem and only $400 + 8n$ for one decomposed with a partial-order task-hierarchy. We start reducing the total problem size when $n > 4$ rooms.

4.2 The Breakfast Task

We demonstrate a decomposition of a breakfast task that uses three variables and stochastic actions. A robot must learn to co-locate a bowl, milk and cereal on a table, eat the cereal, replace the milk in the refrigerator and move the bowl to the sink to be washed up. We represent the kitchen as a 2D 100 location grid-world in which the refrigerator, cereal cupboard, table and sink have separate locations. The state-space is defined by a vector giving the location of the bowl (100 locations), the milk (100 locations) and the cereal (101 locations, one being in the stomach). There are 12 primitive actions allowing each of the objects to be moved Up, Down, Left, Right, and one action to eat the cereal. The actions moving each object are stochastic in the sense that the location of an object at the next step is 80% as intended and 20% as if moved in another direction with equal likelihood. The reward per action is -1 and 1000 for successful completion.

Partial-order HRL constructs the task-hierarchy for the cereal subtask in Figure 1 (top) and finds the abstract policy that is equivalent to the center part of the partial-order plan, Figure 1 (bottom). There are only two abstract states with one multi-tasking action each. The reinforcement learning specification for the original representation of the problem requires a table size of 13,130,000 compared to a partial-order task-hierarchy decomposition of 6,515. Figure 5 shows the improvement in performance. The results including the time to explore and build the subtasks for the partial-order hierarchical reinforcement learner.

5 Future Work and Conclusion

Our contention is that this type of decomposition is common in real world tasks and that robots need to decompose the environment in this way to effectively solve everyday chores. For example in assembly operations the order of construction of components parts themselves may not matter. When writing a paper the order of preparation of the figures is not important, etc.

Future work envisages automatic decomposition when subtasks are interdependent with conflicting or cooperative goals to achieve near optimal performance. The multi-tasking action $\bigoplus$ will then need to arbitrate the decision as to which action to execute within a subtask so as to best compromise between subtask goals. Related research that implements a run-time choice between (possibly) conflicting child subtasks includes W-Learning [14], modular Q-Learning [15], Q-Decomposition [16] and Co-articulation [17]. The challenge is to extend

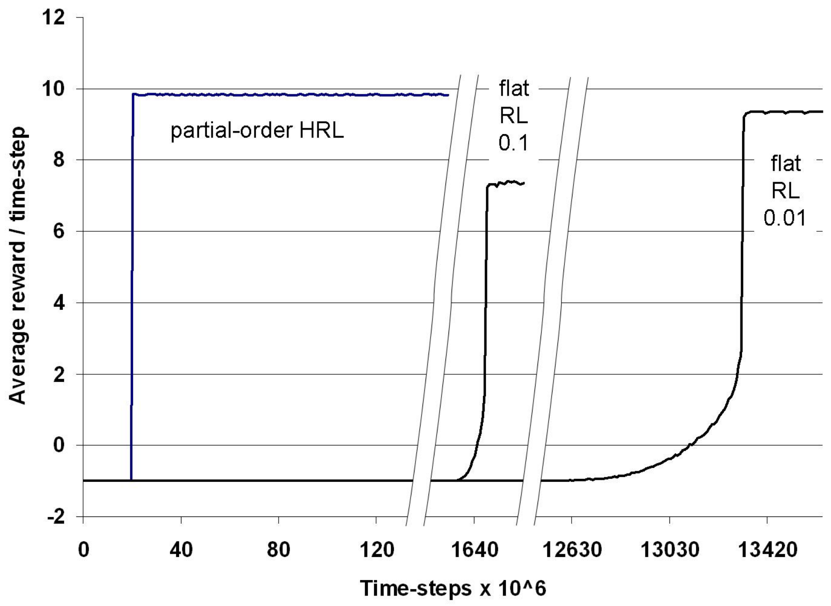

Fig. 5. Breakfast task performance: Partial-order HRL outperforms the "flat" RL by orders of magnitude in learning time and in the level of performance. Two learning rates of 0.1 and 0.01 were used for the "flat" RL.

the automatic discovery of multi-tasking actions to include conflicting subtask choices and continuing (infinite horizon) problems.

Task-hierarchies can be further extended to include abstract concurrent actions $\bigotimes$ by projecting both state and action vectors and modelling the state-action spaces $s^i \times a^j$ for all i, j. The approach would entail abstracting subsets of variables and actions and searching for independent regions that can be executed concurrently. A key difference between a concurrent and a multitasking action is that the former implies parallel execution of primitive actions and the latter does not.

The contributions of this paper include the introduction of partial-order task-hierarchies and their use in automatic problem decomposition and efficient solution of MDPs using a hierarchical reinforcement learning approach. Given that in the real world subtasks tend to be nearly independent, this representation is anticipated to help scale reinforcement learning significantly.

Acknowledgements

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

1. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, Upper Saddle River, NJ (1995)
2. Polya, G.: How to Solve It: A New Aspect of Mathematical Model. Princeton University Press (1945)
3. Turchin, V.F.: The Phenomenon of Science. Columbia University Press (1977)
4. Simon, H.A.: The Sciences of the Artificial. 3rd edn. MIT Press, Cambridge, Massachusetts (1996)
5. Barto, A., Mahadevan, S.: Recent advances in hiearchical reinforcement learning. Special Issue on Reinforcement Learning, Discrete Event Systems Journal **13** (2003) 41–77
6. Dietterich, T.G.: Hierarchical reinforcement learning with the MAXQ value function decomposition. Journal of Artificial Intelligence Research **13** (2000) 227–303
7. Thrun, S., Schwartz, A.: Finding structure in reinforcement learning. In Tesauro, G., Touretzky, D., Leen, T., eds.: Advances in Neural Information Processing Systems (NIPS) 7, Cambridge, MA, MIT Press (1995)
8. Digney, B.L.: Emergent hiearchical control structures: Learning reactive hierarchical relationships in reinforcement environments. From Animals to Animats 4: Proceedings of the fourth international conference on simulation of adaptive behaviour. (1996) 363–372
9. McGovern, A., Sutton, R.S.: Macro-actions in reinforcement learning: An empirical analysis. Amherst technical report 98-70, University of Massachusetts (1998)
10. Şimşek, O., Barto, A.G.: Using relative novelty to identify useful temporal abstractions in reinforcement learning. In: Proceedings of theTwenty-First International Conference on Machine Learning (ICML 2004). (2004)
11. Hengst, B.: Discovering hierarchy in reinforcement learning with HEXQ. In Sammut, C., Hoffmann, A., eds.: Proceedings of the Nineteenth International Conference on Machine Learning, Morgan-Kaufman (2002) 243–250
12. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, Massachusetts (1998)
13. Harel, D.: Statecharts: A visual formalism for complex systems. Science of Computer Programming 8 (1987) 231–274
14. Humphrys, M.: Action selection methods using reinforcement learning. In Maes, P., Mataric, M., Meyer, J.A., Pollack, J., Wilson, S.W., eds.: From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior, MIT Press, Bradford Books (1996) 135–144
15. Karlsson, J.: Learning to Solve Multiple Goals. PhD thesis, University of Rochester (1997)
16. Russell, S., Zimdars, A.: Q-decomposition for reinforcement learning agents. In Proc. ICML-03, Washington, DC (2003)
17. Rohanimanesh, K., Mahadevan, S.: Coarticulation: an approach for generating concurrent plans in markov decision processes. In: ICML '05: Proceedings of the 22nd international conference on Machine learning, New York, NY, USA, ACM Press (2005) 720–727