Learning Recursive HTN-Method Structures for planning

Qiang Yang and **Rong Pan** and **Sinno Jialin Pan**
Dept. of Comp. Sci. & Eng., Hong Kong University of Science & Technology, Kowloon, Hong Kong

Abstract

HTN planning is one of the most effective planning methods in AI. However, designing the HTN-decomposition methods is a very difficult task which has been achieved mainly by humans. It would therefore be desirable to design automated learning methods to acquire these decomposition methods from observed action sequences. In this work, we explore how to apply model-based clustering in order to construct task decomposition hierarchies and summarize a database of action sequences. We present a probabilistic model for unsupervised learning of HTN methods from action sequences. Based on this model, we introduce a novel two-pronged approach by simultaneously learning a Markov model for action segment clusters from action sequences and then learning an action parameter model for recognizing tasks. These models are integrated together to construct action clusters. Then, an abstraction algorithm is applied to extract variables from the action parameters in each cluster to obtain succinct HTN methods. We introduce evaluation metrics for this approach, and test the algorithm in a logistics planning domain.

Introduction

HTN planning applies a divide-and-conquer strategy by taking advantage of the inherently hierarchical structure in planning tasks and application domains (Wilkins 1988; Currie & Tate 1991; Nau *et al.* 2003). Theoretical studies (Erol, Hendler, & Nau 1994) have been conducted on the formalization of HTN planning systems. A key component of a HTN planning system is a set of decomposition methods, whereby a high-level task is reduced to a set of ordered lower level subtasks. HTN methods, consisting of task descriptions as well as how tasks can be decomposed into sequences of subtasks and actions, embody much domain knowledge that encodes the designers' vast problem solving experience. In the past, much of this knowledge has been provided by humans. In this paper, we explore how to automatically acquire the decomposition structure of HTN methods from observed action sequences.

There are three critical learning problems in acquiring HTN methods. The *first problem* is how to acquire the *logical* relationship between a high-level task and its low-level actions for a task decomposition method in terms of preconditions and goals. This problem has been addressed in part by novel machine learning methods given in (Ilghami *et al.* 2005), which learn preconditions and goals of a method from experts' execution traces. A *second problem* is how to learn the preconditions and post-conditions of primitive actions in STRIPS or more sophisticated PDDL action models. Several learning-based techniques have been developed for solving this problem (Amir 2005; Wang 1995; Gil 1994; Blythe *et al.* 2001; McCluskey, Liu, & Simpson 2003; Yang, Wu, & Jiang 2005). The *third problem* in HTN-method learning is how to acquire the *decomposition structures* of HTN methods from observed action traces when each input example consists of *multiple* subsequences of actions for *multiple* tasks. Our objective is to determine which subsequence should belong to which task, a problem of segmentation. This problem is seldom addressed in literature (Reddy & Tadepalli 1997; Nejati, Langley, & Könik 2006), and is the focus of this paper. To make the problem description more realistic, we further assume that we do not have the observed states achieved by the low-level actions. To simplify our discussion, in this paper we refer to this problem as one of *learning HTN-method structures*. When no ambiguity exists, we simply refer to it as the HTN-method learning problem. The output of our solution is a set of pairs, consisting of action sequences and high-level tasks achieved by these sequences in a hierarchical structure.

The HTN-method learning problem can be stated as follows. As input, we are given a set of observed plan traces that consist of sequences of action names and the associated parameters. We are also given a set of high-level tasks or goals that each of these plans are aimed at achieving. We do not assume that we know the states that hold between these actions; in fact, knowing some of these states (partial observation) can help us in improving the accuracy and efficiency of learning, but we do not assume that they must exist. Our objective is to learn a set of HTN methods, where each decomposition method specifies a task and a sequence of low-level actions that can achieve this task. These decomposition methods can also be recursive in nature, so that one can repeatedly apply them to give rise to finer and finer action specifications. As an example, from a collection of plan traces for learning a decomposition method of a non-primitive task *transport-person*$(?person\ ?city2)$, such as *board(John*$, p, c1)$, *transport-aircraft*$(p, c2)$, *debark(John*$, p, c2)$. ..., we wish to learn a disjunctive and recursive HTN-method structure, as shown in Figure 1.

HTN-method learning is a challenging problem. First, we do not make observable intermediate-state requirements. To be sure, the learning problem would be easier if we knew the states just before and after each action, because we can draw

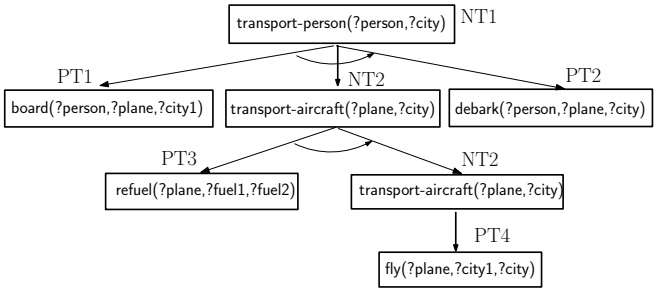

Figure 1: An HTN method example.

upon such strong constraints to relate actions and tasks together. However, in reality, what is observed before and after each action may just be partially known, or even unknown. For example, this may be the case when we were to automate the operation of a Web service by recording the data processing operations of human operators. In our methodology, the training plans can be obtained through monitoring devices such as sensors and cameras, or through a sequence of recorded commands through a computer system such as UNIX domain. Second, the possible assignment of subsequences of actions to high-level tasks constitute a huge search space due to the uncertainty in assigning actions to tasks. In each plan trace, any action such as boardplane(plane0 city1) may be related to any of a set of tasks. It is this uncertainty that causes difficulties for a naive enumeration of potential subsequences for the methods. Finally, an HTN methods may be recursive in nature. How to learn a recursive task-network structure is a challenging problem, because it involves both the recursive decomposition structure and the parameters that are part of the action specification.

It is thus intriguing to ask whether we can approximately learn an HTN-decomposition method model if we are given a set of recorded action signatures and task specifications. In this paper, we take a first step towards answering this question by presenting an algorithm known as *HLS*(HTN Learning System). *HLS* proceeds in two phases. In phase one, *HLS* applies model-based clustering algorithms to find subsequences of actions from the example plan traces for each high-level task. In doing so a set of domain constraints are applied, as well as a statistical learning algorithm that is based on the expectation-maximization (EM). In phase two, *HLS* learns a set of recursive methods from the clusters. The generated HTN methods can then be passed on to human designers for fine-tuning.

Background and Objective

In this paper, we follow the terminology of (Erol, Hendler, & Nau 1994) which defines *Ordered Task Decomposition* in which, at each point in the planning process, the planner has a totally ordered list of tasks to accomplish. An HTN domain consists of the following:

- T is a list of tasks. Each task, which has a name and zero or more arguments (variable symbols), can be either *primitive* (PT) or *non-primitive* (NT). A non-primitive task must be decomposed into a list of subtasks while a primitive task can be carried out directly.
- M is a collection of methods, each of which is a triple $m = (NT, DEC, P)$, where NT is a non-primitive task, DEC is a totally-ordered list of tasks called a decomposition, and P is the precondition of the decomposition.
- O is a collection of operators, each of which is also a triple $o = (PT, DEL, ADD)$, where PT is a primitive task, DEL and ADD are the deleting list and adding list of state after taking the operation o.

An HTN planning process decomposes high-level tasks in an *initial task network* into smaller and smaller subtasks until the task network contains only *primitive tasks* (operators). The decomposition of each task into subtasks uses a *method* from *domain description*.

In this paper, we focus on learning the DEC component of the HTN methods. Suppose we are given the following input:

- A set of plan traces *PLANS*, each of which is represented by a sequence of actions $plan_i = (a_{i_1}, a_{i_2}, \ldots, a_{i_n})$, where i_n is the length of the plan trace $plan_i$. For different plan trace, i_n could be different.
- An initial task network of each plan trace. (Lotem, Nau, & Hendler 1999), given as an ordered list of *non-primitive* tasks decomposed at the top level of the task network, $NT_{i_1}, NT_{i_2}, \ldots, NT_{i_m}$. These are also the high-level goals to be achieved.

HLS: HTN Learning System

Learning the method structure is a two-phase process. In the *first phase*, we perform clustering on the given plan examples. In particular, for each action sequence and its associated non-primitive tasks, we predict a subsequence of actions in the plan example, that can achieve a task. We do this for all plan examples, after which we obtain a cluster of action subsequences for each task. In the *second phase*, we build a succinct summary for each cluster. This summary corresponds to a task signature and HTN method, consisting of a task name and parameter list, and a sequence of subtasks used to accomplish the high-level task. The collection of tasks and their methods is the HTN method structure.

Phase 1: Clustering to Associate Action Subsequences with Tasks

The objective for the first phase of the algorithm is that for each task, we shall induce an HTN Task Cluster Model (HTCM) T from the training examples. At each level of the tree structure, a non-primitive task is represented as a hidden variable, while the actions are considered as observations. These structures can be used to decompose the action

sequences into clusters, each of which is associated with a set of subsequences of actions.

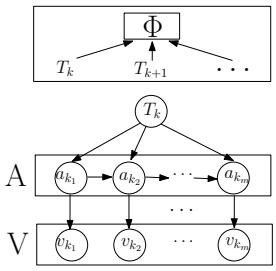

Figure 2: The HTN Task Cluster Model (HTCM).

One example of an HTCM is shown in Figure 2. To fully represent the HTCM, we need to know the joint probability of the hidden factor T_k that represents a task, an action sequence A, and a set of parameters V. The joint probability corresponding to HTCM can be computed as follows.

$$P(T_k, A, V) = P(T_k) \times P(a_{k_1}|T_k) \times P(v_{k_1}|a_{k_1}) \times \prod_{i=2}^{m} \left\{ P(v_{k_i}|a_{k_i}) \times P\left(a_{k_i}|a_{k_{i-1}}, T_k\right) \right\}. \quad (1)$$

According to Eq.(1), in order to represent the HTCM structure and the probabilities, we need to know three compute the joint probability for the tree. In the following, we present two models based on EM algorithm to learn them. We first build Markov models for the action sequences to learn the probabilities $P(a_{k_1}|T_k)$ and $P(a_{k_i}|a_{k_{i-1}}, T_k)$ of Eq.(1). We then build parameter models for the actions and task parameters to learn $P(v_{k_i}|a_{k_i})$ and applying an EM algorithm to learn $P(T_k)$ and parameters of the two models mentioned above. Finally, we show an HTN-method generator based on the learned models.

In the input data, each training example plan is associated with a set of tasks to start with. A difficulty associated with the clustering phase is that, for each action or subsequence of actions in an example plan, we do not know which task is its associated task. No label information is given to us up front. For example, this is equivalent to the following problem: given a sequence of symbols $a_1, a_2, \ldots a_n$ and its associated task labels t_1, t_2, associate $t1$ with a subsequence of a_i's with a task label t_j. The problem is difficult because we do not know which subsequence should belong to t_1 and which other should belong to t_2. Fortunately, for our planning problem, we have two sources of information. One source is the parameter list, which gives us indication on which subsequences are more likely to be associated with a task. Another source is the order information inherent in both action sequences and the tasks in each example.

To build the clusters, we apply an expectation-maximization (EM) algorithm (Dempster, Laird, & Rubin 1977) as shown in Table 1. In this algorithm, initialization is done in Steps 1-3, where the algorithm assigns actions in each plan example to a task by matching the parameters between the actions and the task. Subsequently, in Steps 4-11, the EM algorithm enters a loop consisting of an M-step and an E-step. In the M-Step (Steps 5-7), we re-build the Markov model and the parameter model. In the E-Step (Steps 8-10), we re-assign actions to tasks based the mixture models obtained so far that consist of the Markov models and parameter models. The stop criterion is when the sum of the differences between the current Markov models and the ones in the previous loop is less than a pre-defined threshold.

Algorithm 1 EM Algorithm for HTCM($PLANs, T$)

Input:
$PLANs$ is a set of plan traces $\{plan_i, i = 1, 2, \ldots, N\}$. Each plan trace is an action sequence $plan_i = (a_1^{(i)}, a_2^{(i)}, \ldots, a_{n_i}^{(i)})$ which achieves a set of tasks $\{t_1^{(i)}, t_2^{(i)}, \ldots\ t_{m_i}^{(i)}\}$.;
$T = \{t_1, t_2, \ldots, t_M\}$ is a set of task names.

Output:
The Markov Model M of each task t_k in T.

Steps:
1: **for all** $plan_i$ **do**
2: Initially assign action $a_j^{(i)}$ to a task in light of the number of matches of their parameters, and the output is an action subsequence (keep the order of the original plan trace) of each task;
3: **end for**
4: **repeat**
5: **for all** $t_k \in T$ **do**
6: collect all the subsequences belonging to it (the output of step 1) and build the Markov model and the parameter model;
7: **end for**
8: **for all** $plan_i$ **do**
9: assign action $a_j^{(i)}$ to a task in light of the Markov Markov model and the parameter model of the corresponding task;
10: **end for**
11: **until** a stop condition is satisfied
12: **Return** the final Markov Model of each task .

Building the Markov Models In Steps (1-3) and Steps (5-7) of the EM algorithm, we model an action subsequence for a task using a Markov model. This model consists of the following components

- A set of Q states, $S = \{S_1, S_2, \ldots, S_Q\}$, where Q is the number of actions in the domain;
- The initial state probability distribution is $\pi = \{\pi_i\}$, where $\pi_i = P(q_1 = S_i)$, $1 \leq n \leq Q$.
- The state transition probability is $Mat = \{m_{ij}\}$, where $m_{ij} = P(q_{t+1} = S_j|q_t = S_i)$, $1 \leq i, j \leq Q$.

In the notations above, S_i denotes an action name in the domain. q_t indicates the action state at the t-th action of a task.

Based on these definitions, a Markov model is a pair (π, Mat). We can build such a Markov model for each task. We follow (Rabiner 1989) to build the Markov models.

Example 1 An example plan trace is given below, where the initial task is decomposed into three totally ordered non-primitive subtasks: *transport-aircraft*$(plane1, city1)$, *transport-person*$(person1, city2)$ and *transport-person*$(person2, city1)$. For each subtask, we don't know which sub-action-sequence can achieve it. We will use this example through the paper to explain our algorithm. The action sequence in this example is: *fly*$(plane1, city3, city1, fl2, fl1)$, *board*$(person1, plane1, city1)$, *fly*$(plane1, city1, city2, fl1, fl0)$, *debark*$(person1, plane1, city2)$, *board*$(person2, plane1, city2)$, *refuel*$(plane1, fl0 fl1)$, *fly*$(plane1, city2, city1, fl1, fl0)$, *debark*$(person2, plane1, city1)$.

Now consider the transport-person method in this example. In this problem, the action states are board, fly, debark and refuel. It's easy to compute $\pi = (0.4, 0.2, 0, 0.4)$ and

$$Mat = \begin{pmatrix} 0 & 0.67 & 0 & 0.33 \\ 0.3 & 0 & 0.7 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

Here $\pi_1 = 0.4$ means the task begins with the action *board* with probability 40% . $a_{12} = 0.67$ and $a_{14} = 0.33$ mean that the action fly more likely follows action board than the action *refuel* does because $0.67 > 0.33$; $a_{13} = 0$ means no probability is from *board* to *debark*. **End of Example 1**

Building Parameter Models Step 6 of the HTCM algorithm builds a parameter model for the probability $P(v_{k_i}|a_{k_i})$ of Eq. (1). We use parameter-matching as a heuristic to strengthen our prediction that an action belongs to a task cluster, and use a Naive Bayes (NB) classifier trained on each task to determine whether an action belongs to the task, although other classifiers can also be used. To do this, we generate training examples with input attributes and class labels. First, we construct input features for the classifier. In this problem, there are many methods to extract features for the classifier. Here we present a simple and effective approach. If the action achieves the task, the class label is one. Otherwise, it is zero. For example, given the task *transport-person*$(x1, x2)$ and the action *debark*$(y1, y2, y3)$, we construct six features $X(x1? = y1, x1? = y2, x1? = y3, x2? = y1, x2? = y2, x2? = y3)$. "? =" is a boolean operator which outputs 1 if its left hand side equals to its right hand side, and output 0 in the opposite case. Second, we construct class labels for the feature vectors according to relation between the action and the task.

Consider an example for generating training samples from the training sample in **Example 1**, which corresponds to a "transport-person–debark" classifier. The feature vector of the pair (*transport-person*$(person2, city1)$, *debark*$(person2, plane5, city1)$) is $(1, 0, 0, 0, 0, 1)$ and class label is 1 as the $debark$ action achieves the task. Another pair (*transport-person*$(person2, city1)$, *debark*$(person4, plane5, city3)$) is $(0, 0, 0, 0, 0, 0)$ and class label is zero for the *debark* action does not achieve the task, which achieving task is *transport-person*$(person4, city3)$ (see Example 1). We can also see that the first and sixth constructed features can tell whether the two actions achieve the *transport-*$person$ task or not although they have the same action name "*debark*".

The Markov model and the parameter models are then used together in Step 6 to determine whether each subsequence of actions belongs to the cluster of a particular task. Once the clusters are determined in Step 7, the algorithm reassigns each action a_j to a task that maximizes the probability $P(T_k, A, V)$ in Eq. (1). As an example, each action sequence in the training data is partitioned into subsequences that are associated with tasks. For example, Figure 3 shows a partition induced from Example 1. This process iterates till the assigned clusters converge to a stable state.

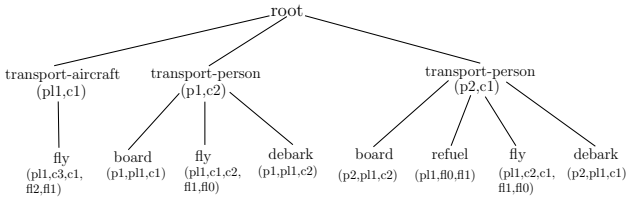

Figure 3: A partitioning of an action sequence by the tasks accomplished by the plan.

Phase 2: HTN-Method Structure Generator

Once the clusters are built using the EM algorithm, we obtain different Markov models for different tasks. In this subsection we present an algorithm to abstract the recursive methods for each task based on its Markov model. The intuition of our algorithm is that in each iteration we find the most frequent action subsequences as a pattern, and treat the pattern as a candidate for an HTN method. We then generate a new non-primitive action to represent the method and replace the "old" action subsequence in the entire training data by the non-primitive action. After obtaining a "new" subplan trace for each task, we rebuild the Markov models respectively. Applying this method repeatedly, we can extract hierarchical HTN methods.

Consider the following example on how to extract the method of the subtask "transport person". The subplan traces for achieving the subtask are shown in Figure 4.

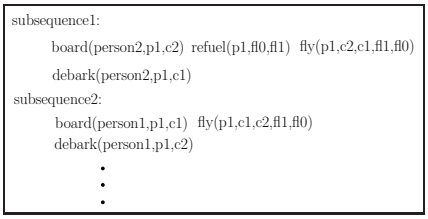

Figure 4: Example plan traces for achieving the task "transport person."

Initially, based on the Markov probability transition matrix learnt from HTCM, we found that

$P(Action(\textit{fly})|Action(\textit{refuel}, Task(\textit{transport-person}))) > \theta$, where θ is a user-defined frequency threshold. We also find that $P(Action(\textit{fly})|Action(\textit{refuel}), Task(\textit{transport-person}))$ is very similar to $P(Action(\textit{refuel})), Task(\textit{transport-person})$, which means that $Action(\textit{fly})$ can be the base case of the recursive method with a high probability. Thus, we pick these two actions as the bottom layer of the hierarchical method and pick $Action(\textit{fly})$ as the base case for the recursion. Subsequently, we use a new action named $NT1$ to replace the two-action sequence in the training dataset where $Action(\textit{fly})$ occurs, and rebuild the subplan traces (shown in Figure 5). After this step, we rebuild the Markov

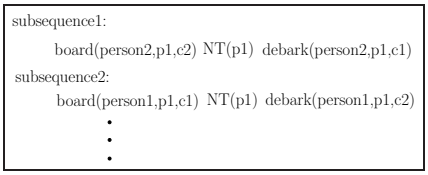

Figure 5: New subplan traces for achieving the task "transport person".

model for these new subplan traces. Finally, we execute step 2 and step 3 repeatedly until the all the elements of Markov transition matrix are lower than θ. This process generates a hierarchical method for a task "transport person", as shown in Figure 7).

Note that our algorithm follows the same spirit as Inductive Logic Programming such as the FOIL system (Quinlan 1990). While both FOIL and our algorithm can both learn recursions from examples, a major difference is that our learning process is guided by statistical patterns rather than by logical patterns. In the future, we will pursue this direction further by integrating the two forms of reasoning.

Experiments

There are two aspects in ensuring the success of the learning algorithm. One is the quality of the HTN methods learned, and another is the speed in executing the EM algorithm. For evaluating the quality of HTCM, similar to (Xu & Muñoz-Avila 2005), we define two performance measures as follows. First, HTN-method-recall = $\frac{\#CMethods}{\#TMethods}$, where #CMethods means number of correct HTN methods correctly covered by the learned HTN methods and #TMethods means the total number of correct methods. Second, HTN-Methods-accuracy = $\frac{\#UMethods}{\#NMethods}$, where #UMethods means number of learned HTN methods accurately cover correct methods and #NMethods means total number of learned HTN methods. If the learned model creates too many methods, the recall measure will be high, and the accuracy measure will be low. In contrast, the HTN-Method-accuracy can be very high while the HTN-Method-recall is very low if the learned model creates just a few methods.

We have done several experiments to verify our algorithm. Due to space limitation, we only show one of them. In our first experiment, we used a well-known ZenoTravel domain for generating the training data. ZeroTravel is one of the domains in the planning competition. The ZenoTravel problem is to transport people form their current locations to their destinations, by use of any available airplanes. In this domain, we have generated 200 action sequences with four actions (fly, board, debark, refuel) and two tasks (transport-aircraft, transport-person) based on methods that are decided by human experts. Action *fly* (?plane ?city1 ?city2 ?fl1 ?fl2) means the ?plane flies from ?city1 to ?city2 and the ?plane's fuel level changes from ?fl1 to ?fl2. *board* (?person ?plane ?city) means the ?person boards onto the ?plane in ?city. debark (?person ?plane ?city) means the ?person debarks from the ?plane in ?city. *refuel* (?plane ?fl1 ?fl2) means the ?plane is refuelled from fuel level ?fl1 to ?fl2. We apply HTCM to the these plan traces and generate HTN methods automatically. In order to evaluate our HTN methods generator, we compare the HTN methods learnt from HTCM with the HTN methods which are given by the human experts. The action sequences in testing set are all labeled by a graduate student for generating the performance measures. The HTN methods created by HTCM are called the learned HTN methods and the ones generated by the human is called the correct HTN methods (this is the ground truth) see Figure 1. We ran our experiments on a Pentium IV PC with 1G memory and 2.2 GHz CPU using Windows XP.

The experimental results are calculated by averaging the results on 10 different randomly-generated the set of plan traces. We show the performance results of recall and accuracy in Figure 6. The X-axis of the figure represents the number of plan traces used for training and the Y-axis represents the HTN-method-recall rate and HTN-method-accuracy in the range [0 1]. We can see that when the number of the training plan traces increases, the HTN-method-recall and HTN-method-accuracy of the learned model increase and converge quickly at 40. This implies we can learn the HTN methods automatically from only 40 plan traces. The HTN methods learned from our models. We also found that the generated HTN method for the task "transport person", which is shown in Figure 7, is similar to the ones designed by human experts, as shown in Figure 1.

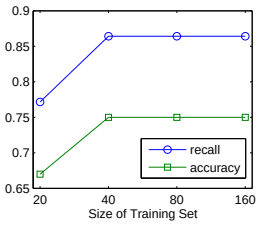

Figure 6: HTN-method-recall and HTN-method-accuracy vs. different numbers of training plan traces.

In Figure 8, we show the learning time with on sizes of the training set. From the figure, we can see that when the number of training plan traces is 40, the CPU time has a relatively small value. This implies our approach is very efficient.

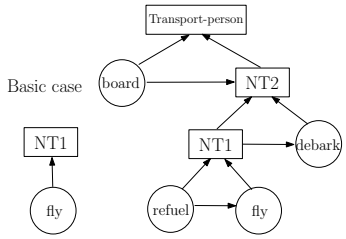

Figure 7: The generated methods for the task "transport person."

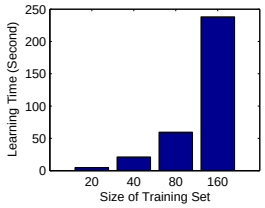

Figure 8: CPU time used for learning.

Conclusion and Further Work

In this work, we explored how to apply model-based learning to construct task decomposition hierarchies for HTN planning from a set of action sequences. We introduce a novel two-phased approach, by simultaneously learning a Markov models and parameter models from the action sequences and learning to cluster the action subsequences according to the high-level tasks. These clusters are used to construct HTN methods recursively. We have shown through several experiments that this approach is effective in learning HTN methods both accurately and efficiently. In the future, we will explore how to learn all parts of HTN methods together, as well as to conduct more tests on the benchmark planning domains, and more evaluations when the domains become more complex. Furthermore, how to determine the threshold θ in each iteration is also our interesting work.

Acknowledgement

We are supported by Hong Kong RGC Research Grant 621606.

References

[Amir 2005] Amir, E. 2005. Learning partially observable deterministic action models. In *Proceedings of the 19th Intl' Joint Conference on Artificial Intelligence (IJCAI'05)*, 1433–1439.

[Blythe *et al.* 2001] Blythe, J.; Kim, J.; Ramachandran, S.; and Gil, Y. 2001. An integrated environment for knowledge acquisition. In *Intelligent User Interfaces*, 13–20.

[Currie & Tate 1991] Currie, K., and Tate, A. 1991. O-plan: the open planning architecture. *Artificial Intelligence* 52:49–86.

[Dempster, Laird, & Rubin 1977] Dempster, A. P.; Laird, N. M.; and Rubin, D. B. 1977. Maximum likelihood from incomplete data via EM algorithm. *Journal of the Royal Statistical Society Series B* 39:1–38.

[Erol, Hendler, & Nau 1994] Erol, K.; Hendler, J.; and Nau, D. S. 1994. Htn planning: Complexity and expressivity. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. AAAI Press.

[Gil 1994] Gil, Y. 1994. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Eleventh Intl Conf on Machine Learning*, 87–95.

[Ilghami *et al.* 2005] Ilghami, O.; Munoz-Avila, H.; Nau, D. S.; and Aha, D. W. 2005. Learning preconditions for planning from plan traces and htn structure. In *Proceedings of the International Conference on Machine Learning (ICML)*. AAAI Press.

[Lotem, Nau, & Hendler 1999] Lotem, A.; Nau, D. S.; and Hendler, J. A. 1999. Using planning graphs for solving htn planning problems. Menlo Park, CA, USA: American Association for Artificial Intelligence.

[McCluskey, Liu, & Simpson 2003] McCluskey, T. L.; Liu, D.; and Simpson, R. 2003. Gipo ii: Htn planning in a tool-supported knowledge engineering environment. In *The International Conference on Automated Planning and Scheduling (ICAPS03)*.

[Nau *et al.* 2003] Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. Shop2: An htn planning system. *Journal of Artificial Intelligence Research* 20:379–404.

[Nejati, Langley, & Könik 2006] Nejati, N.; Langley, P.; and Könik, T. 2006. Learning hierarchical task networks by observation. In *ICML*, 665–672.

[Quinlan 1990] Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine Learning* 5:239–266.

[Rabiner 1989] Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE* 77(2):257–286.

[Reddy & Tadepalli 1997] Reddy, C., and Tadepalli, P. 1997. Learning goal-decomposition rules using exercises. In Fisher, D. H., ed., *ICML*, 278–286. Morgan Kaufmann.

[Wang 1995] Wang, X. 1995. Learning by observation and practice: An incremental approach for planning operator acquisition. In *International Conference on Machine Learning*, 549–557.

[Wilkins 1988] Wilkins, D. E. 1988. *Practical planning: extending the classical AI planning paradigm*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

[Xu & Muñoz-Avila 2005] Xu, K., and Muñoz-Avila, H. 2005. A domain-independent system for case-based task decomposition without domain theories. In Veloso, M. M., and Kambhampati, S., eds., *AAAI*, 234–240. AAAI Press / The MIT Press.

[Yang, Wu, & Jiang 2005] Yang, Q.; Wu, K.; and Jiang, Y. 2005. Learning action models from plan examples with incomplete knowledge. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling*, 241–250. AAAI Press.