From Task Definitions and Plan Traces to HTN Methods

Chad Hogg
Lehigh University

Introduction

Hierarchical Task Network (HTN) planning is an important, frequently studied research topic in artificial intelligence. Researchers have reported work on its formalisms and applications (Erol, Hendler, & Nau 1994; Smith, Nau, & Erol 1998; Nau *et al.* 2005). In HTN planning, complex tasks are decomposed into simpler tasks until a sequence of primitive actions is generated. HTN planning is frequently studied because it is analogous to a common model of human thought and because it has allowed impressive gains in execution time when compared to classical planners.

Despite these advantages, a major hurdle for the use of HTN planning is the need for an HTN domain description. In fact, a controversy in the AI planning research community surrounds the recent efficiency gains obtained with HTN planning because the domain descriptions sketch the underpinnings of the solutions. Therefore, it has been argued that a significant knowledge engineering effort is required to obtain such domain descriptions. A domain description is a collection of knowledge constructs describing the target domain. In HTN planning, a domain description consists of the action model and the task model. The action model encodes knowledge about valid actions or primitive tasks changing the world state. The task model encodes knowledge about how to decompose tasks into subtasks, and is the part of the domain description that has been argued to be difficult to obtain. Given the large interest in HTN planning, it is surprising that little research has been done on learning task models. The bulk of research involving planning and learning has focused on search control knowledge (Zimmerman & Kambhampati 2003).

We present HTN-MAKER (Hierarchical Task Networks with Minimal Additional Knowledge Engineering Required), an offline and incremental algorithm for learning task models. HTN-MAKER receives as input a collection of plans generated by a STRIPS planner, an action model, and a collection of task definitions, and it produces a task model. When combined with the action model, this task model results in an HTN domain model that may be used by an HTN planner to solve problems in the domain. An extended version of this paper has been submitted to the Planning and Learning workshop at ICAPS-07, which includes a description of the learning algorithm and empirical evaluation.

Related Research

Learning task decompositions means eliciting the hierarchical structure relating tasks and subtasks. Existing work on learning hierarchies elicits a hierarchy from a collection of plans and from a given action model (Choi & Langley 2005; Reddy & Tadepalli 1997; Ruby & Kibler 1991). A particularity of the existing work on learning task models is that the tasks from the learned hierarchies are the same goals that have been achieved by the plans. Reddy and Tedepally's 1997 X-Learn, for example, uses inductive generalization to learn task decomposition constructs, which relate goals, subgoals, and conditions for applying d-rules. By grouping goals in this way, task models are learned that lead to speed-up in problem-solving. However, it is possible to solve the same problems without the learned task models.

Two recent studies (Ilghami *et al.* 2005; Xu & Munoz-Avila 2005) propose eager and lazy learning methods respectively to learn the preconditions of HTN methods. These systems require as input the hierarchical relationships between tasks and learn only the conditions under which a method may be used. Another recent work by Langley & Choi 2005 learns a special case of HTNs known as teleoreactive logic programs. Rather than a task list, this system uses a collection of Horn clause-like concepts. The means-end reasoning that is tightly integrated with this learning mechanism is known to be incapable of solving some problems that general HTNs are able to solve, such as the register assignment problem.

Work on learning macro-operators (e.g., (Mooney 1988; Botea, Muller, & Schaeffer 2005)) falls in the category of speed-up learning, as do work on learning search control knowledge ((e.g., (Mitchell, Keller, & Kedar-Cabelli 1986; Minton 1998)). Search control knowledge does not increase the number of problems that theoretically can be solved. However, from a practical stand point, these systems increase the number of problems that can be solved because of the reduction in runtime. Other researchers assumed that hierarchies are given as inputs for learning task models. (Garland, Ryall, & Rich 2001) uses interactive elicitation in which the user provides examples showing how to correctly perform a task and annotates other ways to perform the task in the examples.

Another related work is abstraction in planning such as the Alpine (Knoblock 1993) and the Paris (Bergmann &

Wilke 1995) systems. These systems take a concrete plan and generalize it. This allows the reuse of the generalized plan in different problems by instantiating its conditions. These systems require both an action model and an abstraction model that indicates how to abstract and specialize plans.

Learning Hierarchical Relations From Tasks

We will first specify the problem of extracting hierarchies. In previous work for learning task hierarchies, tasks are goals and therefore the semantics of the learned hierarchies were clear. Given an HTN *H* with a goal *g* at the top level, the plan *P* obtained by collecting the actions in the leaves of *H* must achieve *g* to be correct. That is, one can examine the plan, regardless of the hierarchy, to determine if it is correct. This is not the case in general HTN planning. Informally, a plan is correct if an HTN exists that decomposes the top-level task(s) of the problem such that the HTN entails the plan. The top-level tasks represent complex goals that may not be in the vocabulary of the preconditions and effects of the actions in the plans. This means that the only way to verify if a plan is correct is by finding an HTN that entails it. This poses a problem for defining the kinds of tasks that are given in the task taxonomy so that the semantics of the resulting hierarchy unambiguously relates to the input problem-solution plan pairs.

Task Definitions

To address this problem, we have adopted the definition of tasks from process models. Loosely speaking, a process is the means by which tasks are accomplished via a series of actions or operations. In particular, we chose the task-method-knowledge (TMK) variant of process models. In TMKs, **tasks** indicate what they accomplish by stating their preconditions and effects. Task semantics are the following: if the preconditions are true in the state of the world and the task is accomplished, the effects must be true in the resulting world state. The task definitions used as input for the hierarchy learning problem consist of a collection of tasks in this form. These tasks form the nonprimitive tasks of the domain, while the heads of the operators in the action model form the primitive tasks in the domain. Note the distinction between tasks, which are provided and specify what should be accomplished, and methods, which are learned and specify how to accomplish a task.

Learning Problem

The **task model learning problem** is defined as follows: given a collection of task definitions, a collection of STRIPS problems, a collection of plans solving these problems, and the action model used to generate these plans, obtain a task model. Under these preconditions and given a learned task model, one can check if a plan P **correctly** solves an HTN planning problem where $t_1 \ldots t_n$ are the tasks to achieve, and S is the initial state. To do this, one checks if the preconditions of t_1 are satisfied in S and if the effects of t_1 are satisfied in a state S_1. The state S_1 is obtained by executing the plan P_1 on S, where P_1 is the plan entailed by the portion of the HTN that accomplishes t_1. One can continue by checking task t_2 starting from S_1, and so forth for the remaining tasks.

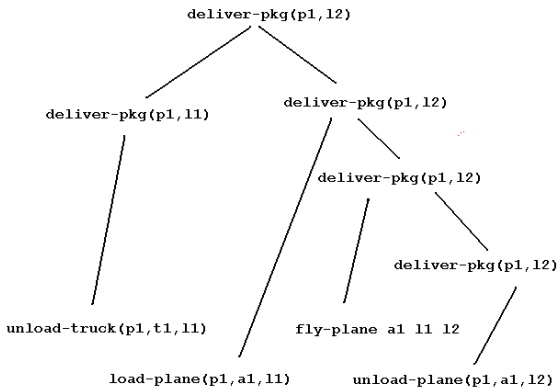

Figure 1: Example of HTN obtained by HTN-MAKER

The HTN-MAKER Algorithm

The HTN-MAKER algorithm traverses forward through a STRIPS plan, generating the new state after each action by applying it to the previous state. For each substitution of variables such that the current state includes all effects of a task it is possible to learn a set of methods: one that encapsulates the previous operator, another for the previous two operators, and so on. The subtasks of these learned methods are the encapsulated operators or previously learned methods and the preconditions are determined by regressing the task effects through the subtasks.

Example

Figure 1 exemplifies a resulting HTN for the logistics-transportation domain. The initial state in this case consists of a package *p1* in a truck *t1* at an airport *l1* that contains an airplane *a1*, and the goal is to deliver the package to a different airport *l2*. The plan consists of four actions: *unload-truck(p1,t1,l1)*, *load-plane(p1,a1,l1)*, *fly-plane(a1,l1,l2)*, and *unload-plane(p1,a1,l2)*.

Suppose that there is a single task, *deliver-pkg(?p,?l)*, with preconditions that *?p* be a package and *?l* be a location, and effects that *?p* be at *?l*. After the first operator, package *p1* has been delivered to location *l1*. Thus, HTN-MAKER will learn a method for solving this task bound to these constants. The operator *unload-truck(p1,t1,l1)* produces the effect *at(p1,l1)*, so it will be selected as a subtask. The learned method will be applicable when the types of variables are correct (from the task preconditions), and the package is in a truck that is at the destination (from the preconditions of the operator). In the next two states, there are no valid instantiations of the task effects.

In the final state, the package *p1* has been delivered to location *l2*. A recursive series of methods is learned. The first delivers a package that is in an airplane at the destination by

unloading the airplane. The next delivers a package that is in an airplane at the wrong location by flying to the destination, which must be an airport, and then delivering. The third requires that the package be at an airport that is not the destination and that contains an airplane, and proceeds by loading the package into airplane and then continuing to deliver. The final first delivers to an airport, and then from there to the final destination.

An HTN planner presented with this initial state, goal, and collection of methods might build the same hierarchical structure from the top task down to the primitive actions. With a different initial state or goal, an HTN planner might use pieces of this structure integrated with other methods learned from other problems.

Open Questions And Future Work

While we have been able to produce good results in the logistics-transportation domain, these do not translate well to the blocks-world domain. In all cases the soundness of the learned domain description in terms of producing only valid STRIPS plans is guaranteed[1], but it is often far from optimal. Specifically, there are two significant difficulties. The first is the very large number of methods that will be learned and that must be considered by an HTN planner using the domain description. The second, more serious problem is the possibility for the planner to use methods in an infinitely recursive manner.

The general question to be studied is the appropriate level of generality. A domain description that is too general allows infinite recursion and erodes the advantages of HTN planning over classical planning. A highly specific domain description is less likely to be capable of solving new problems and will require a much larger set of methods than should be necessary. Finding appropriate techniques for retaining generality while making the learned domains more like those a domain expert would write remains a challenging problem, and I expect it to be the majority of my dissertation.

Acknowledgments

This research was in part supported by the National Science Foundation (NSF 0642882) and the Defense Advanced Research Projects Agency (DARPA).

References

Bergmann, R., and Wilke, W. 1995. Building and refining abstract planning cases by change of representation language. *Journal of Artificial Intelligence Research* 53–118.

Botea, A.; Muller, M.; and Schaeffer, J. 2005. Learning partial-order macros from solutions. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS-05)*. AAAI Press.

Choi, D., and Langley, P. 2005. Learning teleoreactive logic programs from problem solving. In *Proceedings of the Fifteenth International Conference on Inductive Logic Programming*. Springer.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. Htn planning: complexity and expressivity. In *AAAI'94: Proceedings of the twelfth national conference on Artificial Intelligence (vol. 2)*, 1123–1128. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Garland, A.; Ryall, K.; and Rich, C. 2001. Learning hierarchical task models by defining and refining examples. In *Proceedings of the First International Conference on Knowledge Capture*, 363–391.

Ilghami, O.; Munoz-Avila, H.; Nau, D.; and Aha, D. W. 2005. Learning approximate preconditions for methods in hierarchical plans. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Knoblock, C. 1993. *Abstraction Hierarchies: An Automated Approach to Reducing Search in Planning*. Norwell, MA: Kluwer Academic Publishers.

Minton, S. 1998. *Learning Effective Search Control Knowledge: an Explanation-Based Approach*. Ph.D. Dissertation, Carnegie Mellon University.

Mitchell, T.; Keller, R.; and Kedar-Cabelli, S. 1986. Explanation-based generalization: A unifying view. *Machine Learning* 1.

Mooney, R. J. 1988. Generalizing the order of operators in macro-operators. *Machine Learning* 270–283.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Munoz-Avila, H.; Murdock, J. W.; Wu, D.; and Yaman, F. 2005. Applications of shop and shop2. *IEEE Intelligent Systems* 20(2):34–41.

Reddy, C., and Tadepalli, P. 1997. Learning goal-decomposition rules using exercises. In *Proceedings of the Internation Conference on Machine Learning (ICML-97)*.

Ruby, D., and Kibler, D. F. 1991. Steppingstone: An empirical and analytic evaluation. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 527–531. Morgan Kaufmann.

Smith, S. J. J.; Nau, D. S.; and Erol, K. 1998. Control strategies in htn planning: theory versus practice. In *IAAI '98: Proceedings of the tenth conference on Innovative applications of artificial intelligence*, 1127–1133. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Xu, K., and Munoz-Avila, H. 2005. A domain-independent system for case-based task decomposition without domain theories. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*. AAAI Press.

Zimmerman, T., and Kambhampati, S. 2003. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine* 73–96.

[1]The proof of correctness is omitted here, but follows from the way preconditions are effects are regressed through the learned methods