LHTNDT: LEARN HTN METHOD PRECONDITIONS USING DECISION TREE

Fatemeh Nargesian and Gholamreza Ghassem-Sani
Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
nargesian@ce.sharif.edu, sani@sharif.edu

Keywords: AI Planning, hierarchical task network planning, machine learning, decision tree, domain knowledge.

Abstract: In this paper, we describe LHTNDT, an algorithm that learns the preconditions of HTN methods by examining plan traces produced by another planner. LHTNDT extracts conditions for applying methods by using decision tree based algorithm. It considers the state of relevant domain objects in both current and goal states. Redundant training samples are removed using graph isomorphism. Our experiments, LHTNDT converged. It can learn most of preconditions correctly and quickly. 80% of our test problems were solved by preconditions extracted by ¾ of plan traces needed for full convergence.

1 INTRODUCTION

Hierarchical Task Network (HTN) planning is a promising and applicative research topic in Artificial Intelligence. The basic idea of HTN was first developed in mid-70s (Sacerdoti 1975; Tate 1977). Its formal underpinnings were developed in mid-90s (Erol, Hendler, and Nau 1996). An HTN planning problem consists of an initial state, a set of tasks to be performed as problem goal, and a domain description. HTN domain description contains a set of operators as primitive tasks (they can be performed directly in the domain during execution time) and a set of methods describing possible ways of decomposing tasks into subtasks and subtasks into primitive tasks. Each method has a precondition. In order to apply a method on a planning state, its precondition must be satisfied in that state. Planning is done by applying methods to decompose non-primitive tasks into subtasks, and applying operators to primitive tasks to produce actions. Planning ends when all of the tasks mentioned in the goal set are satisfied, then the planner has found a solution plan; otherwise the planner will need to backtrack and try other applicable methods and actions that are not considered yet (Ilghami, *et al.* 2005).

HTN is a configurable planner whose domain knowledge is provided by a human domain expert to achieve satisfactory performance. Therefore, such planners' functionality depends on domain-specific problem solving knowledge to be accurate. It should be born in mind that the designer of a domain for a configurable planner generally has many valid alternative ways of specifying the domain, and it is well known that the exact form of the domain can have a large impact on the efficiency of a given planner. Even if a human designer can identify some of the complex manner in which the tasks and operators in a domain description interact, he will likely be faced with tradeoffs between efficiency and factors such as compactness, comprehensibility and expressiveness. Consequently, there are obvious advantages to a planner that can evolve its domain theory via learning. Learning is the process of using past experiences and percepts to improve one's ability to act in the future. The extensive survey and analysis of research work related to machine learning as applied to planning reveals that machine learning methods can be used in learning and improving planning domain theory (Zimmerman and Kambhampati 2003). In this paper, we discuss a learning algorithm for evolving HTN planning domain automatically.

In recent years, several researches have reported work on integrating learning methods and HTN planning. An example is a system called HICAP, developed by Munoz-Avila *et al.* 1999, which uses planning in military environment. HICAP integrates SHOP, a hierarchical planner, (Nau, *et al.* 1999) and a case-based reasoning (CBR) system called NaCoDAE (Aha and Breslow 1997). Learning HTN domain means eliciting the hierarchical structure

relating tasks and subtasks. Existing work on learning hierarchies, extracts hierarchy from a collection of plans having primitive operators' descriptions (Ruby and Kibler 1991; Reddy and Tadepalli 1997; Choi and Langley 2005). The idea is that the tasks are the same as goals that have been achieved by the plans. Reddy and Tedepally, in X-Learn system, uses inductive generalization to learn task decomposition constructs, named D-rules, which relate goals, sub-goals and conditions of goal decomposition.

Research on learning macro-operators (Korf 1987; Mooney 1998; Botea, Muller, and Schaeffer 2005) falls in the category of learning hierarchical structures in planning domains. ALPINE (Knoblock 1993) and PARIS (Bergmann and Wilke 1995) systems are good examples of using abstraction in planning. Knoblock presented a completely automated approach for generating abstractions of problem solving using a tractable, domain-independent algorithm. The inputs of this system are the definition of a problem space and the problem to be solved; and the output is an abstraction hierarchy that is tailored to particular problem. Two recent studies (Ilghami *et al.* 2005; Xu and Munoz-Avila 2005) propose eager and lazy learning methods respectively to learn the preconditions of HTN methods. Ilghami, in CaMeL, assumes that method definitions are available and uses Candidate Elimination Algorithm to extract methods' preconditions from plan traces. In HDL (Ilghami, Nau, and Munoz-Avila 2006), there is no prior information about the methods and it learns HTN domain description by examining plan traces produced by another HTN problem-solver. Another recent work, by Langley and 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 most recent work, by Hogg 2007, presents HTN-MAKER, an offline and incremental algorithm for learning task models. HTN-MAKER receives as input a collection of plans generated by a STRIPS planner (Fikes and Nilsson 1971), an action model, and a collection of task definitions; and produces a task model. When combined with the action model, this task model results in an HTN domain model.

Here, we introduce LHTNDT (Learn HTN using Decision Tree), an algorithm that uses a decision tree based learning method for learning preconditions of HTN methods. It is assumed that system has the knowledge of general structure of decomposing tasks into subtasks. But this knowledge is incomplete in case that it does not have sufficient information about where to use the method to be successful and efficient. LHTNDT learns conditions for efficient application of methods by doing analysis on plan traces that are known to be successful or unsuccessful for certain problem instances. The preconditions are shown in the formalism of a decision tree.

The paper is organized in 5 sections. Section 2 overviews the inputs to the learning algorithm and its output. Section 3 discusses the learning algorithm. Section 4 reports empirical results of applying learning method on a planning domain. Finally section 5 draws conclusions and describes future work.

2 INPUTS AND OUTPUTS OF LHTNDT

Inputs to the learning algorithm are annotated plans. We use a set of optimal plan traces, which contain not only the correct methods used for a planning problem, but also information about possible decisions that could be made while this plan was being generated. This form of input is often preferable because it will result in faster and more accurate learning, because plan traces contain much more information about the domain being learned than plans (Ilghami *et al.* 2005). The annotated plan traces are in the form of trees whose nodes are the states that the planner has passed through in order to achieve a goal. For each node we consider not only the methods that are on the path, to the goal state but also other applicable methods that have caused failure in planning or optimal planning.

The annotated plan traces are in the form of trees whose nodes are the states that the planner have passed through in order to achieve the goal.

The decision tree based learning algorithm is used for extracting complete domain knowledge for HTN. By using decision trees, planner is not obliged to test all method preconditions during test time. In the proposed algorithm, one decision tree is generated for each domain method. The target attribute of theses trees specify whether to use the corresponding method or not.

Thus, the output of LHTNDT is certain decision trees as preconditions of the domain methods. According to the structure of decision trees, the precondition learned for HTN by LHTNDT is a disjunctive normal form of sentences.

```
Inputs: S={S1, …, Sn}, a set of world states
        M={m1, …, mn}, the set of operators in HTN domain (D)
        Π={ Π1, …, Πn}, a set of plan traces
        (Ii, Gi, D), planning problem
        I: Initial State
        G: Goal State
Output: DT, a set of decision trees, each of them represents the precondition of
a method

LHTNDT(S, I, G, M, Π)
DT=0, Samplesi=0
FOR each plan trace Πi ∈ Π
     FOR each node n in Πi, whose si ∈ S except I and G
          FOR each method mi ∈ M considered in n
               Let samp be a training sample
               samp.CurrentState = n.State
               samp.GoalState = Πi .GoalState
               IF applying mi to n results in a non-leaf node
           Samp.Type = "Positive"
               ELSE
                    Samp.Type = "Negative"
               ExtractImportantObjects (samp, mi)
               IF (AddSample (Generalize (samp, mi), Samplesi, mi)))
              MakeDTTrainingSample (samp, Samplesi)
FOR each method mi in M
     LearnDT(Samplesi, mi, DT)
RETURN DT
```

Figure 1: The LHTNDT algorithm.

3 LHTNDT ALGORITHM

Pseudo-code of LHTNDT is given in Figure 1. For learning the preconditions of applicable methods of a state, we consider both the current and goal states of the plan. The attributes of the decision trees are the combination of domain predicates and all objects of the domain. These attributes are doubled when they are considered for both appearing in the goal and current states. The values that these attributes can take are *True, False* and *Don't Care*. The third value provides the system with the ability to check just the attributes that are important as preconditions of the method, during the test phase. LHTNDT starts by defining and initializing decision trees. Then for each node in each given plan trace, it produces a training sample for each applicable method. If applying the method on a node results in a non-goal leaf node, it reveals that using the method in that situation resulted in failure or a non-optimal plan. Thus, the sample extracted from it should be of type *Negative*. Otherwise, the produced sample is considered as *Positive* sample. The sample is added to training set if it does not have common structure with other existing samples. Finally, LHTNDT learns certain decision trees for domain methods. The algorithm converges when learned decision tress for methods converge. In other words, hypothesis spaces for preconditions converge to a fixed point. The subroutines of LHTNDT are as follows:

- **ExtractImportantObjects(*samp*, m_i)**, is a function that iteratively extracts the objects of the domain that are somehow related to method m_i. The main idea is inspired by Ilghami *et al.* 2005. The relative situation of domain objects in the current state and also their ultimate situation in the goal state specifies whether to apply the method or not. Considering the objects that are directly or indirectly related to the objects on which method is applied, assists the system not to learn the facts that are not useful. The function starts by initializing relevant objects to the set of objects that have appeared in a predicate of current state with at least one of the objects in the arguments of method m_i. Then, in each subsequent iteration, it alternatively processes goal state and current state and adds those remaining objects that are in a predicate in which one of the objects in the relevant objects

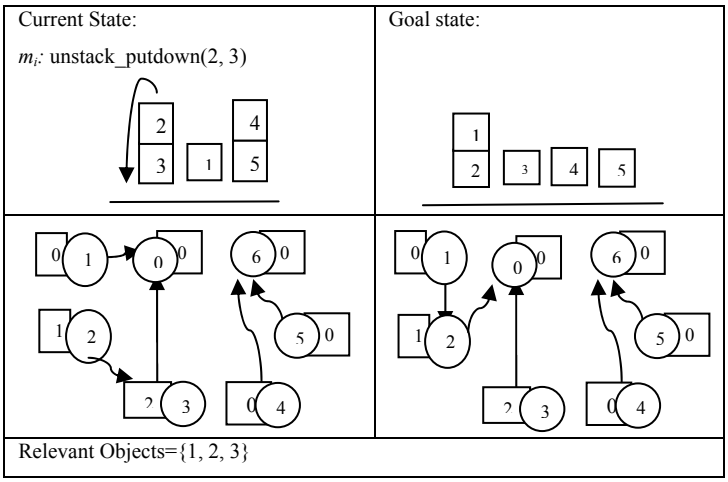

Figure 2: Graphs for a training sample in Blocks World domain (node 0 is considered as table, node 6 is defined for specifying non-relevant objects).

set appeared as an argument. These iterations continue until no new objects are added to the relevant objects set.

- **Generalize(*samp*, m_i)**, make the generalized form of the method and also the current and goal state mentioned in the training sample *samp*.
- **AddSample(*samp*, $Samples_i$, m_i)**, where $Samples_i$ is the set of all unique training samples for method m_i. This function creates a graph for each sample. The graph has two components, one for the initial state and the other for the goal state. The nodes of the graph are domain objects, named as various variables. Each node is labeled by a number indicating whether the corresponding object is a relevant object. The edges of the graph are labeled with the name of the predicate that two nodes are appeared in. The nodes that are arguments of m_i, are labeled with their argument order in the method. These graphs are also the generalized form of the samples. An example of the graph produced for a sample is depicted in figure 2. In order to prevent from adding a sample with the same structure as existing samples, we perform the test of graph isomorphism between the graph of *samp* and the graphs of other samples. If *samp* is isomorphic with none of the samples in $Samples_i$, *samp* is a new sample and should be added to the $Samples_i$ set. Thus, the function returns *True*, otherwise it returns *False*. The idea of using graphs for extracting new samples can also be used for the predicates of more than 2 arguments. The algorithm should be modified in such a way to group and merge nodes that are involved in a common method and adjoin node groups.
- **MakeDTTrainingSample(*samp*, $Samples_i$)**, assigns value to the attributes of decision tree in order to make a training sample. It maps sample graph nodes to the generalized objects in attributes by matching the node of the first argument of the method of *samp* with the first generalized object defined in attributes and continuing this matching by traversing the graph and mapping generalized object of attributes and newly-met objects of the graph. For each of the attributes, if none of the objects in the attribute is mapped with one of the objects in sample relevant object set, the value of the attribute is set to *Don't Care*. Otherwise, if the ground predicate made out of the attribute and its mapped objects is among initial state or goal state (according to the attribute name), the value of the attribute is *True* and in the case it is not, the value is *False*.
- **LearnDT($Samples_i$, m_i, *DT*)**, creates a decision tree for each of the methods according to training sample in $Samples_i$.

4 EMPIRICAL EVALUATION

One of the challenges in any learning algorithm is how to define a criterion to evaluate its output. In this section, we explain some points about implementing LHTNDT algorithm. Then, we discuss our experiments to figure out how many plan traces are needed to converge and also the precision

of the planning domain that LHTNDT has learned by just ¾ of plan traces needed for convergence.

As mentioned in previous section, LHTNDT extracts training samples from certain annotated optimal plan traces. Thus, LHTNDT, unlike previous systems which require plan traces produced by an expert or a planner working with a hand-tailored domain, does not need any priori knowledge. For accuracy of training samples, we needed all optimal solution plans of a problem. So that, we firstly used an iterative deepening Depth First Search (DFS) algorithm modified for planning. But this planner often failed in domains with many objects because of their time and space complexity. In order to fix this problem, we modified HSP_a*, which is an optimal temporal planner, a member of heuristic search planners. HSP* produces parallel optimal plans. Since optimal parallel plans are not certainly optimal sequential plans, optimal sequential plans are extracted from HSP_a* outputs and plan traces are created for them. We have assumed a method as a sequence of actions (a two level hierarchy of tasks). Therefore, the operators for planning are action sequences (macro-operators) that are defined as methods' structure. The domain used in our experiments is the blocks world, with 4 operators and 3 methods. It is based on the block-stacking algorithm discussed in (Gupta and Nau 1992). For training the system, we generated 4 sets of random problems with 3, 5, 6, and 10 blocks using BWSTATES program (Slaney and Thiébaux 2001). The uniqueness of the problems is checked by making a graph for each of the problems and testing its isomorphism relationship with other problem graphs. The nodes of the graphs are domain objects and the edges are labeled with predicate names and a number showing which state this predicate belongs to: the initial or goal states. Figure 3 compares the number of plan traces LHTNDT needed to converge.

Although more information is provided in plan traces of larger problems, the number of plan traces needed by LHTNDT to converge smoothly increases as more objects are defined in the domain. The reason is that the number of attributes and the situations that should be learned (current and initial states) increase. It should be also born in mind that a plan trace contains more negative samples than positive ones and this may cause later convergence.

An HTN planner is implemented for testing the preconditions learned by LHTNDT. This planner acts just like SHOP (Nau, *et al.* 1999), but in case that more than one method is recommended, by decision tree, to be applied, a random method is chosen. Often, there is not enough information in the input or there are not enough training samples to derive an optimal output. We used just ¾ of problems needed for convergence of LHTNDT for training and tested the learned preconditions on a random test set. In the diagram depicted in figure 4, the precision of method preconditions learned by ¾ problems needed for convergence is shown. Although the planner could not find a plan for all test problems, but it can be seen that LHTNDT can learn most of method preconditions even before full convergence, because it tries to extract as much information as it can form input plan traces. The precisions are high because we have taken into account the condition of objects both in the current state and goal state.

5 CONCLUSIONS AND FUTURE WORK

LHTNDT is an algorithm that learns method preconditions for HTN planner. Its input consists of annotated optimal plan traces produced by another planner (HSP*). LHTNDT takes into account the structure of current and goal state for learning conditions of applying a method. Diverse structure of training samples are extracted by checking the graph isomorphism among created graphs for samples. In our experiments in blocks world domain, although LHTNDT needed various plan traces to converge, according to the results for precision of planning domain before full convergence of preconditions, it learned most of necessary conditions of methods relatively quickly. Regarding the fact that no structurally repetitive training sample is considered for learning and also the relative state of relevant domain objects in both current and goal states are explored, it can be claimed that LHTNDT produces correct preconditions for HTN planning domain.

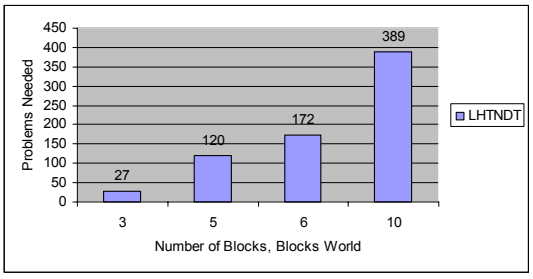

Figure 3: Number of plan traces needed to converge.

A decision tree represents the preconditions of domain methods as disjunctive normal form of sentences. We believe that some rules can be extracted from the decision trees. Therefore, axioms can be used in presenting the preconditions. If the existence of recursion is verified in a method precondition decision tree, the rules will be written in such a way that the rules that are extracted from smaller domains can be used for larger domains. LHTNDT algorithm learns preconditions of HTN methods. Our future work will include developing techniques to learn HTN methods' structures from plan traces produced by another planner.

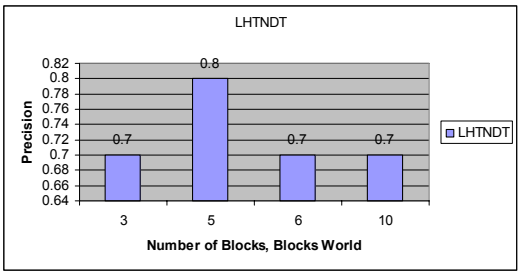

Figure 4: Precision of method preconditions learned by ¾ of plan traces needed for full convergence.

REFERENCES

Aha, D. W.; and Breslow, L. A. 1997. Refining Conversational Case Libraries. In *Proceedings of the Second International Conference on Case-Based Reasoning*, pp. 267-278. Providence, RI: Springer Press/Verlag Press.

Bergmann, R.; and Wilke, W. 1995. Building and Refining Abstract Planning Cases by Change of Representation Language. *Journal of Artificial Intelligence Research*, pp. 53-118.

Botea, A.; Muller, M.; and Schaeffer, J. 2005. Learning Partial-order Macros from Solutions. In *Proceedings of the 15th 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 15th International Conference on Inductive Logic Programming.* Springer.

Erol, K.; Hendler, J.; and Nau, D. S. 1996. Complexity Results for Hierarchical Task-Network Planning. *Annual of Mathematics and Artificial Intelligence* 18(1), pp. 69-93.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence (2)*, pp 189-208.

Gupta, N.; and Nau, D. 1992. On the complexity of Blocks-world Planning. *Artificial Intelligence* 56(2-3): pp. 223-254.

Hogg, Ch. 2007. From Task Definitions and Plan Traces to HTN Methods, *Workshop of International Conference on Automated Planning and Scheduling (ICAPS-07)*.

Ilghami, O.; Nau, D. S.; Munoz-Avila, H.; and Aha, D. 2005. Learning Preconditions for Planning from Plan Traces and HTN Structure.

Ilghami, O.; Munoz-Avila, H.; Nau, D. S.; and Aha. D. 2005. Learning Approximate Preconditions for Methods in Hierarchical Plans. In *Proceedingsof the 22nd International Conference on Machine Learning*, Bonn, Germany.

Ilghami, O.; Nau, D. S.; and Munoz-Avila, H. 2006. Learning to Do HTN Planning, *International Conference on Automated Planning and Scheduling (ICAPS-06)*.

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

Korf, R. E. "Planning as Search: A Quantitative Approach", *Artificial Intelligence,* 33(1), pp. 65-88, 1987.

Mooney, R. J. 1998. Generalizing the Order of Operators in Macro-Operators. *Machine Learning,* pp. 270-283.

Munoz-Avila, H.; McFarlane, D.; Aha, D. W.; Ballas, J.; Breslow, L. A.; and Nau, D. S. 1999. Using Guidelines to Constrain Case-based HTN Planning. In *Proceedings of the Third International Conference on Case-Based Reasoning,* 288-302. Providence, RI: Springer Press.

Nau, D. S.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. SHOP: Simple Hierarchical Ordered Planner. In *Proceedings of the Sixteenth International Joint conference on Artificial Intelligence,* 968-973. Stockholm: AAAI Press.

Reddy, C.; and Tadepalli, P. 1997. Learning Goal-decomposition Rules using Exercises. In *Proceedings of the International Conference on Machine Learning (ICML-97).*

Ruby, D.; and Kibler, D. F. 1991. Steppingstone: An Empirical and Analytic Evaluation. In *Proceedings of the 9th National Conference on artificial Intelligence:* pp. 527-531.

Sacerdoti, E. 1975. The Nonlinear Nature of Plans. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence,* Tiblisi, USSR, pp. 206-214.

Slaney J. K.; and Thiébaux S. 2001. Blocks World revisited. *Artificial Intelligence* 125(1-2): pp. 119-153.

Tate, A.; 1977. Generating Project Networks. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence,* Cambridge, MA, pp. 888-893. MIT Press.

Xu, K.; and Munoz-Avila, H. 2005. A Domain-independent System for Case-based Task Decomposition without Domain Theories. In *Proceedingsof the 20th National Conference on Artificial Intelligence (AAAI-05)*. AAA Press.

Zimmerman, T.; Kambhampati, S. 2003. Learning-assisted Automated Planning: Looking back, tracking Stock, Going Forward. *AI Magazine,* 73-96.