Learning Probabilistic Hierarchical Task Networks from Plan Examples to Capture User Preference

Nan Li
Advisor: Subbarao Kambhampati and Sungwook Yoon
School of Computing and Informatics
Arizona State University
Tempe, Arizona 85281 USA

Abstract

Hierarchical task networks provide an efficient way to encode user prescriptions about what constitute good plans using component methods. However, manual construction of these methods is complex and time consuming. In this paper, we propose a novel approach to learning probabilistic hierarchical task networks that capture the user preference by examining user-produced plans given no prior information about the methods. We introduce a theoretical basis to formally state the schema learning problem. We then make the connection between probabilistic grammar induction and probabilistic hierarchical task network learning, and propose a learning algorithm that shares ideas with probabilistic grammar induction.

Introduction

In recent years, planning systems have been widely used in practical applications ranging from autonomous space vehicles (Muscettola, Nayak, Pell, and Williams, 1998) to robot control (Ghallab & Lauruelle, 1994) to computer bridge (Smith, Nau, & Throop, 1998). Besides generating valid plans, one important requirement to such systems is to generate plans that meet user preference. *Hierarchical task networks (HTN)* (Nau et al., 1999; Wilkins & desJardins, 2001) have received increased attention for providing a flexible way of encoding user intent to control planning. However, generating hierarchical task networks by hand is often difficult and time consuming. Even for the same task, hierarchical task networks need modifications for different users. Although there has been a growing body of work on learning in this framework (Ilghami et al., 2002; Langley and Choi, 2006), to the best of our knowledge, none of them have focused on capturing user preference.

In this paper, we propose an approach that constructs user-oriented probabilistic hierarchical task networks from user-produced plan examples. We assume that all plan examples have the same goal and are generated by users that share the same preference. Each plan consists of a sequence of ground actions. Unlike most of the other learning mechanisms, we assume that hierarchical task networks are completely unknown to the system, which is, unfortunately, usually true in real life. Our objective is to reconstruct the hierarchical task networks associated with the probabilities of being chosen based on the plans provided. Having realized the connection between probabilistic grammar induction and probabilistic hierarchical task networks learning, we developed an expectation-maximization (EM) algorithm inspired by ideas from probabilistic grammar induction systems.

The main contributions of this paper are: 1) We introduce a theoretical basis for formally defining algorithms that learn user-oriented probabilistic hierarchical task networks; 2) We describe the close ties between HTN learning and probabilistic grammar induction; 3) We present a learning mechanism for constructing probabilistic hierarchical task networks.

In the following sections, we begin by briefly reviewing HTN planning and extend classical hierarchical task networks to probabilistic hierarchical task networks to support better encoding of user preference. We then formally define the learning task and state the assumptions we made. Next, we discuss the relations between probabilistic grammar induction and HTN learning. After that, we present an algorithm that acquires user-oriented probabilistic hierarchical task networks. Finally, we examine how our approach relates to earlier work and discuss future research directions.

Background

Hierarchical task networks planning is a generalization to classical planning (Kambhampati et. al, 1998). In addition to primitive actions, a domain description also includes non-primitive actions and a set of reduction schemas, which encode a user's prescriptions by specifying how to reduce non-primitive actions into other actions and/or non-primitive actions. In other words, a plan is acceptable if it is not only able to achieve the goal, but also able to be parsed by the reduction schemas (Barett & Weld, 1994).

For example, in a travel domain as shown in Figure 1, the domain description consists of primitive actions, *Getin, Buyticket, Getout*, and *Hitch-hike*, as well as non-primitive actions, *Gobybus, Gobytrain*, and *Travel*. The reduction schemas state that *Travel* can be reduced to either *Gobybus* or *Gobytrain*. *Gobybus* and *Gobytrain* can be further reduced to primitive actions. Note that *Hitch-hike* is also a way to travel. However, since there is no schema to reduce *Travel* to *Hitch-hike*, the plan *Hitch-hike* will not be considered as a valid plan.

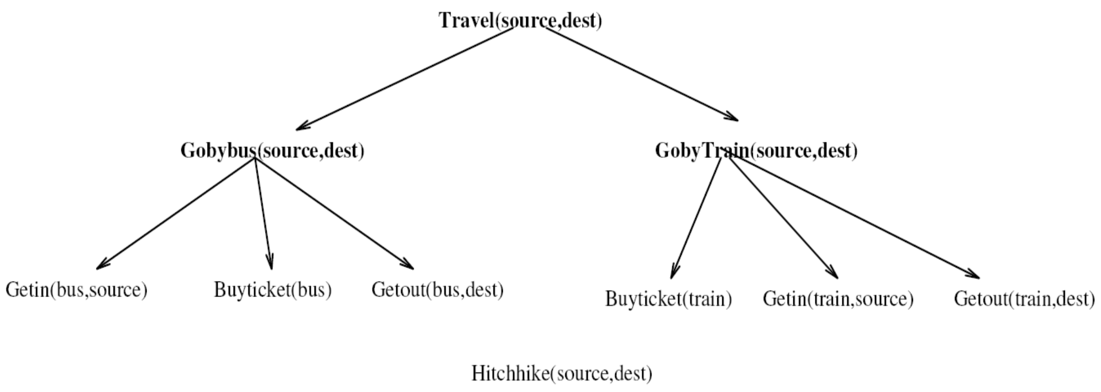

Figure 1: Hierarchical task networks in a travel domain. (Kambhampati et. al, 1998)

The parsability of a plan in hierarchical task networks is deterministic. A plan is either acceptable or not. However, user preference is usually blurry. For instance, although both *Gobybus* and *Gobytrain* are acceptable plans from the above hierarchical task networks. The user may still prefer traveling by train rather than traveling by bus. He/She may take a train 80% of the time, while taking a bus the rest of the time. To capture this kind of preferences, we extend hierarchical task networks to probabilistic hierarchical task networks, where each reduction schema is associated with a probability stating the possibility to get chosen when applicable. In the next section, we will present a formal definition of a constrained version of probabilistic hierarchical task networks.

Problem Definition

In this paper, we focus on a constrained form of probabilistic hierarchical task networks, *ordered probabilistic hierarchical task networks (ordered probabilistic HTN)*. We model an ordered probabilistic HTN domain by a 3-tuple, $\langle A, NA, S\rangle$, where A is a set of primitive actions, NA is a set of non-primitive actions, and S is a set of reduction schemas indexed by non-primitive actions. Each primitive action a_i is a 3-tuple, $\langle PC, OP, EF\rangle$, where PC (precondition) and EF (effects) are first-order boolean formulae, and OP is a operator. Each non-primitive action na_i is associated with a set of reduction schemas. Each reduction schema s_j can be seen as a 5-tuple, $\langle NA, DEC, PC, B, P\rangle$, where NA is a non-primitive action, DEC is an ordered list of primitive or non-primitive actions, PC (the preconditions) is a first-order boolean formula, B is a set of binding constraints on the variables appearing in DEC and PC, and $P(na \rightarrow dec \mid h)$ is a decomposition distribution that represents the probability of decomposing na to dec.

A probabilistic HTN planning problem is a 3-tuple, $\langle I, G, H\rangle$, where I is the complete description of an initial state, G is the description of the goal state, H is an ordered probabilistic HTN domain description described above. An action sequence S is a valid solution to a planning problem if and only if S can be executed from I, the resulting state implies G, and S can generated from G following the reduction schemas in H. An ordered probabilistic HTN domain

Table 1: Ordered probabilistic hierarchical task networks in the travel domain

Primitive actions: $Buyticket, Getin, Getout, Hitchhike$;
Non-primitive actions: $Travel, Gobybus, Gobytrain$;
$Travel \rightarrow 0.2,\ \ Gobybus$
$Travel \rightarrow 0.8,\ \ Gobytrain$
$Gobybus \rightarrow 1.0,\ \ Getin\ \ Buyticket\ \ Getout$
$Gobytrain \rightarrow 1.0,\ \ Buyticket\ \ Getin\ \ Getout$

is said to successfully capture user preference if the distribution of the valid plans generated from the ordered probabilistic HTN forms the same distribution of the user produced plans.

Given the definition stated above, we are ready to describe the input and output specifications. The input of the learning system is a set of plan examples, O, generated by users sharing the same preference. Each plan example o_i is a 2-tuple, $\langle S, G\rangle$, where S is an action sequence, and G is a goal that the plan achieved. The goals of all the plans share the same predicate. The objective of our learning algorithm is to construct an ordered probabilistic HTN domain H that captures user preference based on plan examples o, that is,

$$argmax_H\ p\,(O \mid H) \tag{1}$$

Due to the hardness of learning hierarchical task networks, we first simplified the learning task by ignoring the parameters in the plan traces. This assumption is reasonable in domains such as the travel domain, but does not work with domains like Blocks World, where the action sequences are differentiated between each other by bindings. Second, our algorithm assumes total executability among all actions, which means a plan is an acceptable solution as long as it can be parsed by the reduction schemas. This assumption enables us to focus on learning the structural knowledge embedded in hierarchical task networks.

Based on these two assumptions, the output of the learning algorithm is an ordered probabilistic HTN domain with all PCs and Bs in the reduction schemas S set to be empty. We will sometimes use $na \rightarrow p,\ \ a_1a_2 \ldots a_n$ to denote a reduction schema, where na is a non-primitive action, and

Table 2: Ordered probabilistic hierarchical task networks of Chomsky normal form in the travel domain

Primitive actions: $Buyticket, Getin, Getout, Hitchhike$;
Non-primitive actions: $Travel, A_1, A_2, A_3, B_1, B_2$;
$Travel \rightarrow 0.2,\ \ A_2\ \ B_1$
$Travel \rightarrow 0.8,\ \ A_1\ \ B_2$
$B_1 \rightarrow 1.0,\ \ A_1\ \ A_3$
$B_2 \rightarrow 1.0,\ \ A_2\ \ A_3$
$A_1 \rightarrow 1.0,\ \ Buyticket$
$A_2 \rightarrow 1.0,\ \ Getin$
$A_3 \rightarrow 1.0,\ \ Getout$

$a_1, a_2, \ldots, a_n$ are primitive actions or non-primitive actions. Besides, since all the plan examples aim to achieve the same goal (predicate), the start symbol of the reduction schemas is the goal g. Table 1 shows an example of the ordered probabilistic HTNs in the travel domain. For instance, the schema $Gobybus \rightarrow 1.0,\ \ Getin\ \ Buyticket\ \ Getout$ means *Gobybus* can be reduced to three actions with probability 1.0. More specifically, the reduction schemas learned will be of only two forms,

$$na \rightarrow p,\ \ na_1 na_2 \tag{2}$$

$$na \rightarrow 1.0,\ \ a \tag{3}$$

where na, na_1 and na_2 are non-primitive actions, a is a primitive action, and p is the probability of schema 2 getting selected. Note that ordered probabilistic HTNs of this form are as expressive as the original ordered probabilistic HTNs. Table 2 shows the reduction schemas equivalent to table 1 in the constrained form. More details will be discussed in the next section.

Relating HTN learning to Probabilistic Grammar Induction

After formalizing the syntax of ordered probabilistic HTNs, we realized that the reduction schemas described above form a *probabilistic context free grammar (PCFG)*, where each non-primitive action corresponds to a non-terminal symbol, each primitive action corresponds to a terminal symbol, and each reduction schema is a grammar rule. As is known to all, every context free language can be represented by a context free grammar in Chomsky normal form. It is easy to prove that, every probabilistic context free language can also be generated by a probabilistic context free grammar in Chomsky normal form. Having realized the close connection between HTN and probabilistic context free grammar, we further constrains the format of learned probabilistic HTNs to be of Chomsky normal form as shown in schema 2 and schema 3.

Moreover, we noticed that learning ordered probabilistic HTNs is closely related to the field of grammar induction in probabilistic context free grammar. Grammar induction is to discover common structures embedded in some sequential (sentence) or structured data. Grammar induction on probabilistic context free grammar assumes the embedded

Algorithm 1: $learn$ constructs a set of reduction schemas, S, from the plan examples, O using an EM algorithm until convergence.

```
Input: Plan Example Set $O$.
1  $S$ := $\phi$;
2  $T$ := $\phi$;
3  $S$ := pre-process($O$);
4  while not-converged do
5      forall $o_i \in O$ do
6          $T$ := $T$ + paring($o_i$, $S$);
7      end
8      forall $s_i \in S$ do
9          $sel$ := number-of-times-selected($s_i$, $T$);
10         $tol$ := number-of-times-selected($s_i.na$, $T$);
11         $s_i.p$ := $sel$ / $tol$;
12     end
13 end
14 return $S$
```

common structure is a PCFG. Comparing with HTN learning, the input of HTN learning is a set of plan examples, while the input of grammar induction is a set of sequential data. In both cases, the inputs are sequences of terminal symbols. Moreover, both learning tasks aim to discover a structure embedded in the inputs, and assume the embedded structure is a probabilistic context-free grammar. Last but not the least, the notion of user preference also has its counterpart in grammar induction. First, in HTN learning a plan is acceptable if it can be parsed by HTN schemas, while in grammar induction a sentence is valid if it it is grammatical correct. Second, in HTN learning, different users prefer different kinds of plans, while, in grammar induction, various people have different talking styles.

There has been quite a few work in the area of grammar induction on probabilistic context free grammar (eg. Wang & Acero, 2002; Collins, 1997; Charniak, 2000; Lari & Young, 1990). Lari & Young (1990) developed an expectation-maximization algorithm, *the inside-outside algorithm* to induce grammar. The algorithm takes all possible grammar rules of Chomsky normal form into consideration, and uses the selection probabilities associated with them to compute the probability of generating an observation sequence given the grammar. Then, the algorithm updates the selection probabilities of the grammar rules by the computed probabilities. This process operates in an iterative fashion until convergence.

Learning User-Oriented Probabilistic Hierarchical Task Networks

Inspired by the inside-outside algorithm, we took an approach that employs an EM algorithm, but different from the inside-outside algorithm. The learning algorithm begins by a pre-processing function which constructs an initial probabilistic HTN domain from the example sequences as the start point for the EM algorithm. It then iteratively re-estimates the reduction schemas until convergence. In each

Algorithm 2: $pre - process$ constructs a initial set of reduction schemas, S, from the plan examples, O.

```
Input: Plan Example Set $O$.
1  $S$ := primitive action reduction schemas;
2  forall $o_i \in O$ do
3      if parsable($o_i$, $S$) then
4          continue;
5      else
6          $S$ := $S$ + generate-schema($o_i$, $S$);
7      end
8  end
9  S = initialize-probabilities($S$);
10 return $S$
```

iteration, the algorithm constructs the most probable parsing tree for each plan given the current reduction schemas in the E step. It then updates the estimation of the reduction schemas based on the parsing trees. In this section, we first describe the EM algorithm. Next, we introduce a preprocessing algorithm to provide a better initial estimates. Finally, we discuss properties of the algorithm.

Probabilistic Hierarchical Task Networks Learner

The HTN schema learner starts by guessing a set of reduction schemas S that can generate all plan examples as a staring point. This is done by a pre-process algorithm, which will be discussed in the following section.

Note that for consistency the following constraint must always be satisfied,

$$\sum_{\substack{a_j, a_k \in NA \\ a_i \to a_j\, a_k}} p\left(a_i \to a_j\, a_k \mid H\right) + \sum_{\substack{a_j \in NA \\ a_i \to a_j}} p\left(a_i \to a_j \mid H\right) = 1 \tag{4}$$

This constraint simply means that all non-primitive actions must be either a pair of non-primitive actions or a single primitive actions. Actually, since we assumed that each primitive action is associated with only one reduction schema, a non-primitive action can be reduced to either pairs of non-primitive actions or a single primitive action, but not both. That is to say, one of the two terms in the above equation should be 0.

Since all plan examples can be generated by the target reduction schemas, each plan should have a parse tree associated with it. However, the tree structures of the example plans T are not provided. Therefore, we consider T as hidden variables. We will use T(o, H) to denote the parse tree of a plan example o given the reduction schemas H. The algorithm operates iteratively. In each iteration, it involves two steps, an E step, and an M step.

In the E step, the algorithm estimates the values of the hidden variables T, which, in this case, is are the tree structures associated with each plan example, denoted as

$$p\left(T \mid O, H\right) \tag{5}$$

To do this, the algorithm computes the most probable parse tree for each plan example. Any subtree of a most probable parse tree is also a most probable parse subtree. Otherwise, replacing the original subtree with the most probable parse subtree, we get another parse tree that is of higher probability than the original most probable parse tree. Therefore, for each plan example, the algorithm builds the most probable parse tree in a bottom-up fashion until reaching the start symbol g. For the lowest level, since each primitive action only associates with one reduction schema of the form $na \to a$, the most probable parse trees for them are directly recorded with their only associated reduction schemas of probabilities set to be 1. For higher levels, the most probable parse tree is decided by

$$\begin{aligned} T(o,H) =& argmax_{s,i}\ p\left(s \mid H\right) * p\left(T(o_1,H) \mid o_1, H\right) \\ & \qquad * p\left(T(o_2,H) \mid o_2, H\right), \\ & T(o_1,H), \\ & T(o_2,H). \end{aligned} \tag{6}$$

where o is an action sequence, $a_1, a_2, \ldots a_n$, s is a reduction schema of the form $a_i \to a_j\ a_k$, i is an integer between 1 to n, o_1 is an action sequence, $a_1, a_2, \ldots a_i$, and o_2 is an action sequence, $a_{i+1}, \ldots a_n$. The probability of that parse tree is

$$p(T(o,H)|o,H) = p(T(o_1,H)|o_1,H) * p(T(o_2,H)|o_2,H) \tag{7}$$

where o is an action sequence, o_1 and o_2 are action sequences that correspond to the division of o according to the most probable parse tree. This bottom-up process continues until it finds out the most probable parse tree for the entire plan.

After getting the parse trees for all plan example, the algorithm moves on to the M step. In this step, the algorithm updates the selection probabilities associated with the reduction schemas by maximize the expected log-likelihood of the joint event

$$H_{n+1} = argmax_H \Sigma_T\ p\left(T \mid O, H_n\right) \log p\left(O, T \mid H\right) \tag{8}$$

Probabilities of reduction schemas associated to primitive actions are still 1. For a reduction schema $a_i \to a_j\ a_k$ associated to non-primitive actions, the new probability of getting chosen is simply the total number of times $a_i \to a_j a_k$ appearing in the parse trees divided by the total number of times a_i appearing in the parse trees.

After finishing the M step, the algorithm starts a new iteration until convergence. The output of the algorithm is a set of probabilistic reduction schemas.

Pre-process Algorithm

As with standard EM algorithm, the reduction schemas learned with our algorithm converge toward only local optimum. Moreover, a good starting point for the EM algorithm requires less iterations to converge. Furthermore, considering all possible rules may lead to a huge redundancy in the constructed reduction schemas. Hence, it is essential to find a good initial estimates.

Therefore, instead of starting with random probability values, we designed a pre-process algorithm that obtains a set of reduction schemas S from the example sequences. The pseudo code of the pre-process algorithm is shown in algorithm 2. The reduction schema set S is initialized to contain schemas associated with primitive actions. Then, for each plan example o_i, the algorithm first tries to parse the o_i with the current S. If it succeeds, it will not add any reduction schema to S and move on to the next plan example. If it fails, among all the parse forests, which are partial parses of o_i that can not be further parsed, the pre-process algorithm takes the parse forest that has the fewest number of trees, and generates a set of reduction schemas for the roots of the trees. In order to keep the schemas succinct, the reduction schema generation process prefers to generate balanced and recursive reduction schemas.

After learning the reduction schemas, the pre-process algorithm assigns the probabilities associated with them. Due to the rule shown in Equation 4, for each reduction schema $a_i \rightarrow a_j\ a_k, p$ is assigned to 1 divided by the number of reduction schemas that have a_i as the head. In order to break the symmetry among all reduction schemas, the algorithm adds a small random number to each probability and normalizes the values again. This pre-process algorithm provides a redundant set of reduction schemas to the EM algorithm.

Discussion

Having introduced the learning algorithm, we will discuss several properties of the algorithm in this section. First, since EM algorithms do not guarantee converge to global optimal, we may need to run the learning algorithm multiple times to find reduction schemas of good quality.

Second, the proposed algorithm learns the schema structure not only in the pre-process step, but also in the EM step. Although the EM step does not introduce new reduction schemas, it deletes redundant reduction schema by assigning low or zero possibilities to those schemas.

Third, although our algorithm assumes no prior schema information, it is also capable of acquiring user preference given full or partial schema knowledge by putting the known schemas into the initial candidate methods.

Fourth, in the pre-process algorithm, the slightly asymmetry encoded in the probability values may lead to a large bias on a subset of the reduction schemas, that is, some reduction schemas will be given very small probabilities. A post-process algorithm can be added so that after learning the HTN schemas, the post-process algorithm tries to remove schemas associated with small probabilities as long as the remaining schemas can still express all plan examples.

Finally, the objective of our learning mechanism is to produce the probabilistic HTNs that best describe the user preference. However, this may lead an overfitting problem. By generating exact reduction schemas for each example plan, we will get the reduction schemas that produce only the training example. To solve this problem, we can either limit the total number of non-primitive actions, or extend our algorithm to take advantage of negative plan examples.

Related Work

The main claim of this paper is learning HTN schemas to capture user preference without prior knowledge about schemas. Although there has been considerable work on HTN learning (Ilghami et al., 2002; Langley and Choi, 2006), to the best of our knowledge, most of them have not focused on capturing user intent, and usually require method information given as input. Ilghami et al.'s (2002) work learns domain knowledge also by analyzing successful traces, that contain hierarchical structural information, while the input of our algorithm does not include any structural information. Langley and Choi's (2006) work constructs HTN schemas in the context of problem solving and execution, but again the proposed mechanism depends on partial information of the hierarchical structure. Nejati et al.'s (2006) approach constructs HTN schemas from expert traces based on partial structural information. Moreover, none of the above works have focused on capturing user preference. Other works on building action models (Blythe et al., 2001; Yang et al.,2005; Wang 1995; Oates & Cohen, 1996; Gil, 1994; Shen, 1994; Sablon & Bruynooghe, 1994; Benson, 1995) also learn domain knowledge, but the constructed action model will only contain information about primitive actions. These works essential focus on learning preconditions and effects of primitive actions.

Our framework also incorporates ideas from other research on grammar induction. For example, the E step in the algorithm to build the most probable parse trees bears a clear resemblance to work on parsing algorithm (Collins, 1997; Charniak, 2000). Collin's parser represents parse trees using probabilities of dependencies, while our approach uses reduction schemas to represent parse trees. Charniak's (200) work defines a score function to measure the quality of the parse and returns the parse tree with the highest score. In contrast, our approach does not use any score function.

Concluding Remarks

In summary, we provided a theoretical basis for the learning problem, and showed the connection between HTN learning and grammar induction. We then proposed a probabilistic HTN learning approach that captures user preference from user-produced plan examples with no prior information. The approach employs an expectation-maximization algorithm. Finally, we discussed properties of the proposed algorithm.

There are many possibilities for improvement of this work, which is encouraging. First, we need to carry out experiments to better demonstrate and understand the quality of the learned probabilistic HTNs. We should also compare our algorithm with other learning algorithms, such as the inside-outside algorithm. Second, the proposed EM algorithm does not consider minimizing the size of the reduction schemas. We will extend our algorithm to learn smaller sized reduction schemas by choosing parse trees that are not only of the highest probabilities, but also of smaller sizes in terms of the number of reduction schemas used during parsing. Third, we made several assumptions to simplify the learning task. One interesting extension would be to learn hierarchical task networks with binding information. We will also extend the HTN learner to take executability into consideration. We believe that our HTN learner extends naturally in these directions.

References

Barrett, A., & Weld, D. (1994) Task decomposition via plan parsing. *Proceedings of the Twelveth National Conference on Artificial Intelligence*, (pp. 1117-1122). Seattle, WA.

Benson, S. (1995). Inductive learning of reactive action models. *Proceedings of the International Conference on Machine Learning*, (pp. 47C54). San Fracisco, CA.

Blythe, J., Kim, J., Ramachandran, S., & Gil, Y. (2001). An integrated environment for knowledge acquisition. *Proceedings of Intelligent User Interfaces*. (pp. 13-20).

Charniak, E. (2000). A maximum-entropy-inspired parser. *Proceedings of the European Chapter of the Association for Computational Linguists.*

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. *Proceedings of the Thirty-fifty Annual Meeting of the Association for Computational Linguistics*, San Francisco. Morgan Kaufmman.

Ghallab, M., & Laruelle, H. (1994). Representation and control in lxTet, a temporal planner. *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, Chicago, IL, (pp. 61-67). AAAI Press.

Gil, Y. (1994). Learning by experimentation: Incremental refinement of incomplete planning domains. *Proceedings of the Eleventh International Conference on Machine Learning*. (pp. 87-95).

Ilghami, O., Nau, D. S., Mu noz-Avila, H., & Aha, D. W. (2002). CaMeL: Learning method preconditions for HTN planning. *Proceedings of the Sixth International Conference on AI Planning and Scheduling*, (pp. 131-14). Toulouse, France.

Kambhampati, R., Mali, A., & Srivastava, B. (1998). Hybrid planning for Partially hierarchical domains. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, (pp. 882-888). Menlo Park, CA.

Langley, P., & Choi, D. (2006). Learning recursive control programs from problem solving. *Journal of Machine Learning Research, 7*, (pp. 493-518).

Lari, K., Young, S. (1990). The estimation of stochastic context free grammars using the inside-outside algorithm. *Computer Speech Language, 4*, (pp. 35-56).

Muscettola, N., Nayak, P., Williams, B. (1998). Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence, 103(1-2)*, (pp. 5-47).

Nau, D., Cao, Y., Lotem, A., & Munoz-Avila, H. (1999). SHOP: A simple hierarchical ordered planner. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (pp. 968-973).

Nejati, N., Langley, P., & Konik, T. (2006). Learning hierarchical task networks by observation. *Proceedings of the Twenty-Third International Conference on Machine Learning* (pp. 665-672). Pittsburgh, PA.

Oates, T., & Cohen, P. R. (1996). Searching for planning operators with context-dependent and probabilistic effects. *Proceedings of the Thirdteenth National Conference on Artificial Intelligence*, (pp. 865-868). Menlo Park, CA.

Sablon, G., & Bruynooghe, M. (1994). Using the event calculus to integrate planning and learning in an intelligent autonomous agent. *Proceedings of Current Trends in AI Planning*. (pp. 254-265). IOS Press.

Shen, W. 1994. *Autonomous Learning from the Environment*. Computer Science Press, W.H. Freeman and Company.

Smith, S., Nau, D., & Mitchell, T. (1982). Learning from solution paths, An approach to the credit assignment problems. *AI Magazine 3(2)*, (pp. 48-52).

Wang, X. (1995). Learning by observation and practice: An incremental approach for planning operator acquisition. *Proceedings of Wang, X. 1995. Learning by observation and practice: An incremental approach for planning operator acquisition*. (pp. 549-557).

Wang, Y., Acero, A. (2002). Evaluation of spoken language grammar learning in the ATIS domain. *Proceedings of International Conference on Acoustics, Speech, and Signal Processing.*

Wilkins, D. E., & desJardins, M. (2001). A call for knowledge-based planning. *AI Magazine, 22*, (pp. 99-115).

Yang, Q., Wu, K., & Jiang, Y. (2005). Learning action models from plan examples with incomplete knowledge. *Proceedings of The International Conference on Automated Planning and Scheduling.*