Ontology Oriented Exploration of an HTN Planning Domain through Hypotheses and Diagnostic Execution

Li Jin and Keith S. Decker
University of Delaware
Department of Computer and Information Sciences
Newark, DE 19716, USA
{jin, decker@cis.udel.edu}

Abstract

We present a framework, HTN-Explorer, for case-based exploration of hierarchical task network planning domain oriented by ontological background knowledge. With an existing simple, incomplete model as a seeding model extracted from limited examples of plan solutions, HTN-Explorer explores more comprehensive planning domain knowledge by expanding the seeding model through a discovery circle of hypothesis generation, evaluation and diagnostic execution. HTN-Explorer proposes hypothetical task methods by adapting an existing model for those situations not covered by the original model. As well, hypothetical models are evaluated with heuristics that estimate the plausibility and discoveries of hypotheses. The executions of hypothetical plans based on hypothetical models provide information to diagnosis. This framework provides some desirable functionalities: (1) it automatically explores HTN models by integrating various strategies; (2) it proposes hypotheses for experimental testing based upon their evaluated plausibility and discoveries; (3) it facilitates encoding of background knowledge into the exploration processes. We use a variation of the UM Translog domain to evaluate our approach.

Introduction

Hierarchical task networks (HTNs) are an important, frequently studied approach to solve problems in AI planning research and have recently achieved several notable successes (Nau 2007). A *hierarchical task network* (HTN) planner solves a problem by following task decomposition descriptions to recursively decompose a complex task into simpler tasks until the tasks can be accomplished by actions directly. HTN planning was first presented in the mid-1970s (Sacerdoti 1975), and its formalisms and properties were well studied in the mid-1990s (Erol, Nau, and Hendler 1994). Over the past decade, many planning systems based on HTN decomposition (e.g. SIPE (Wilkins 1985), O-PLAN (Currie and Tate 1991), and SHOP (Nau et al. 2005)) have made successful achievements in the practical applications, such as the Mars Rovers (Estlin et al. 2003) and Bridge Baron (Smith, Nau, and Throop 1998).

Despite the achievements of HTN planning, there still exists a significant challenge, i.e. the difficulty of acquiring complete domain knowledge required by a planner to solve a problem. In many application domains, it is difficult or impossible to acquire accurate and complete HTN models hard-coded by human experts due to multiple reasons, such as the lack of necessary experiences or knowledge of the domains, the complexities in the domains, and time or effort consumption. Consequently, researchers have recently shown strong interest in developing algorithms or systems to (semi)automatically learn planning theories from solution cases/examples or from planning and execution experiences (Zimmerman and Kambhampati 2003). However, most of the previous research that learns a planning domain theory from examples and experiences are passive, namely, it depends on examples or previous planning experiences but cannot actively explore new models not represented by or not reasoned from example cases or experiences.

In this paper, we present HTN-Explorer, a general framework that aims at facilitating exploration of an HTN planning domain with minimum human intervention. Our work is motivated by many practical domains in which generally, background knowledge is available, but domain specific task decomposition or action descriptions may be only partially provided by a human or learned by limited example cases. In such a domain, there might be limited plan examples or solution experiences available that do not cover all situations in the real world; thus, only example-driven learning techniques (e.g. case-based learning) might not work very well. We present an approach to learn comprehensive models through hypothesis generation and diagnostic execution.

With an assumption that background knowledge can be represented in ontology, we develop HTN-Explorer as a self-directed automated system that utilizes an ontology to expand an incomplete model by presenting hypotheses with multiple strategies. The plausibility of a hypothesis is evaluated based on the assessment of the strength of its proposing strategy and the underlying computational method. The novelty of a hypothesis is estimated by new preconditions, new kinds of tasks, new constrains or new solutions that the hypothesis may cover or provide. HTN-Explorer is capable of presenting those hypotheses with high evaluations of plausibility and novelty to a human or an experimental system (e.g. a lab robot) for testing. The feedback information will be used to update the original domain theory and will be utilized to generate new hypothetical models.

Copyright © 2010, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

This paper continues with brief introduction of HTN planning formalism. Next, we overview the architecture of HTN-Explorer. After presenting the hypothesis generation strategies implemented in HTN-Explorer, we will then explain the hypothesis evaluation heuristic function and the diagnosis procedures. Then, we demonstrate an empirical case study of exploring a variation of the UM Translog domain (Andrews et al. 1995). Finally, we discuss related works and make conclusions.

Preliminaries

In this paper, we follow the principles of the task decomposition formalism of HTN planning defined in Chapter 11 by Ghallab et al. (Ghallab, Nau, and Traverso 2004). HTN planning uses task decomposition description methods to decompose non-primitive (also called compound) tasks into simpler subtasks. Planning continues the decomposition process until primitive tasks are reached. A primitive task can be accomplished by an action. A plan solution is composed of a sequence of actions that can achieve the high-level tasks of a problem.

Generally, an HTN planning theory is composed of two sets of descriptions, one is a set of operators, the other is a set of methods. An operator describes how a state is changed if the operator is applied to the state when required preconditions are satisfied. A method describes how a compound task is decomposed into subtasks.

An operator is represented by a 4-tuple $O = (name(O), PreC, Add, Del)$, where $name(O)$ is the name of the operator, $PreC$ is a set of preconditions, and Add and Del are the adding list and deleting list that define how to modify the current state s (consisting of a collection of ground atoms) when O is applied to s by adding or deleting atoms in the lists to or from s. A method is formalized as a triple $M = \{T, PreC, SubTs\}$ specifying that T, a non-primitive task, can be achieved by the subtasks $SubT$ when the elements in the precondition set $PreC$ are satisfied. A task is of the form $T(r_1, \ldots, r_n)$, where T is a task symbol, i.e. the name of the task, and $r_1, \ldots, r_k$ are terms.

An HTN planning problem is a 4-tuple $P = (s_0, Ts, Os, Ms)$, where s_0 is the initial state, Ts is the initial sequence of tasks, and Os and Ms are sets of planning operators and methods respectively. A solution for P is a plan consisting of a sequence of actions that can achieve Ts from the initial state s_0. As a ground instance of a planning operator, an action is of the form $a = (name(a), preconditions(a), effects(a))$. An action can accomplish a ground primitive task t in a state s if a is applicable to s when the preconditions of the action are satisfied in s, and a is able to produce the effects (including adding list and deleting list) that can succeed in achieving the goals requested by the task.

We assume that an ontology describing background knowledge of a domain is available. We use predicate to represent the relationships of variables and constants in an ontology, such as *(class ?x C)* indicating that a variable *?x* is of a class type *C* and *(isa C C')* indicating that *C* is a subclass of *C'*. For instance, Figure 1 shows the ontology of the UM Translog domain that is simplified from its original version

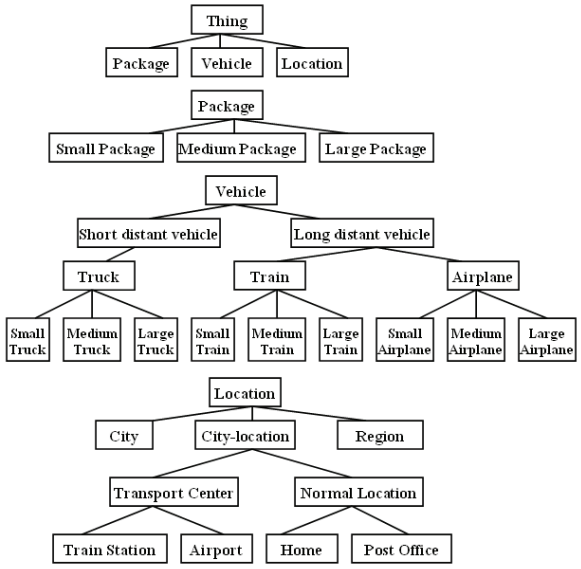

Figure 1: Ontology of the UM Translog domain.

(Andrews et al. 1995) and will be used in this paper to explain our approach. In Figure 1, the relationship that *Small Truck* is a subclass of *Truck* is represented by *(isa Small-Truck Truck)*. And the relationship of class *Small Truck* and its instance *st1* can be defined as *(class st1 Small-Truck)*. We also suppose that a simple incomplete model can be easily hard coded by hand or extracted from limited solution examples. Then the simple model works as a seeding one from which a more comprehensive model can be automatically generated by using some strategies with the aid of ontology knowledge.

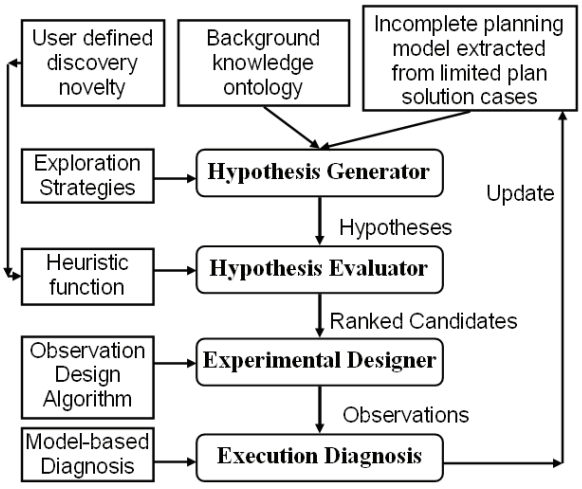

Figure 2: Information flow of HTN-Explorer.

Overview of HTN-Explorer

Figure 2 shows the top-level control of the exploration process in the HTN-Explorer. The process repeats the cycle of hypothesis generation, evaluation, experiment and diagnosis until no hypothesis has a plausible value above a stop criterion.

The HTN-Explorer's input consists of: (1) An initial set of cases of planning problems and their HTN solutions that are used to obtain the initial model to be expanded. The set may be limited; thus, the initial model may be incomplete. (2) Domain-specific background knowledge base stored in an ontology. (3) Domain-specific novelties of discoveries that are defined by a user. The output of HTN-Explorer is an enhanced planning theory that is more comprehensive and more accurate than the original one.

HTN-Explorer consists of the following main components:

- A hypothesis generator that proposes plausible hypothetical HTN methods or operators. It encodes multiple general HTN planning model exploration strategies to propose hypotheses based on the background knowledge organized in an ontology until reaching stopping values of the strategies.
- A hypothesis evaluator that estimates the plausibility and novelty of a hypothesis through a heuristic function. It selects candidates that are worth of the expense of experimental tests. It provides a flexible framework for the usage of specific background knowledge and user defined discoveries.
- An experiment designer that can design observations to be made for a robot or a system or a human that can execute experiments to test hypotheses, make observations, and maintain the records.
- An experimental diagnosis component that interprets the observed results and explains any divergence of the observations from the expected results predicted by the tests. The planning knowledge base is updated by the diagnoses.

Hypotheses Generation

With a set of HTN plan solution cases as input, HTN-Explorer first generates a simple model by abstracting an object with a variable of the deepest class in the ontology that the object belongs to. For the example shown in Figure 3, a method in the right side is generated from the case in the left side.

Then HTN-Explorer uses the following strategies to expand a simple model.

Analogical Expansion

One kind of hypotheses can be generated by modifying a method through replacing an object variable with a similar one in the object's ontology leaf nodes. The similarity between two classes in one ontology is estimated quantitatively as the division of the number of their common ancestors by the sum of the numbers of their individual distinguish ancestors as shown in Equation (1). Hypotheses are created by this method when the similarity between the substituting and original classes is higher than a prefixed value.

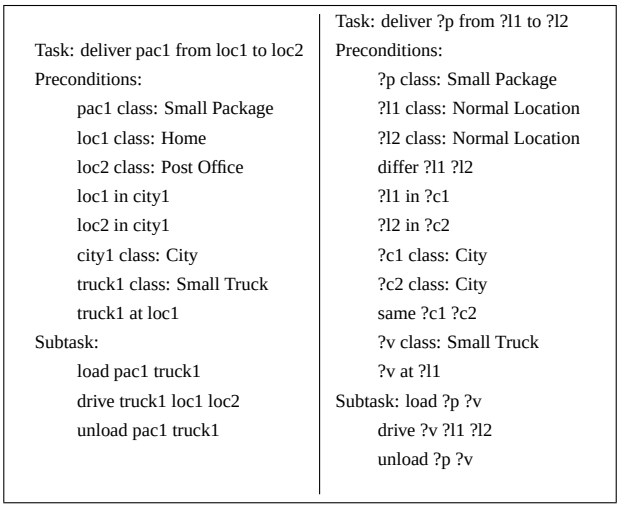

Figure 3: Generating simple model from plan cases.

$$Sim(C_1, C_2) = \frac{|Anc(C_1) \bigcap Anc(C_2)|}{|Anc(C_1) \bigcup Anc(C_2)|} \quad (1)$$

where $Anc(C)$ is a set of all ancestors of C.

For example, based on the ontology shown in Figure 1, *Sim(Small Truck, Medium Truck)*=1.0 and *Sim(Small Truck, Small Train)*=1/5=0.2; thus, *Medium Truck* is decided more similar to *Small Truck* than *Small Train*. As shown in Figure 4, a hypothetical method can be generated by adapting the method in the right side of Figure 3 with modifying the class of *?v* from *Small Truck* to *Medium Truck*. After the hypothesis and testing processes, the method in the right side of Figure 3 will be adapted with all leaf classes in the ontology shown in Figure 1.

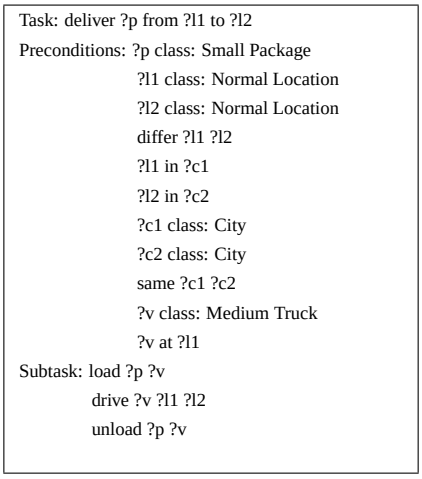

Figure 4: Hypothetical method generated by analogical expansion.

However, hypotheses generated by this strategy may be incorrect. For example, if a hypothetical method can be generated by adapting the method in the right side of Figure 3 with modifying the class of *?p* from *Small Package* to *Medium Package*, then a solution generated by this method will fail when it is executed. Therefore, hypotheses need experimental tests to prove it is correct or not. The testing and diagnosis processes will be discussed in later sections.

Negation

The above strategy expands a seeding model by adapting the model to more class types. There exist other kinds of preconditions that are not related to class types, such as *(same ?c1 ?c2)* in Figure 3. The strategy of negation makes a precondition unsatisfied by replacing it with a negative one; then, the old method will fail under the negative precondition. One way to repair the old method is adding a new subtask that can use an already known method if it exists to achieve the original precondition from the negative one. Thus, a new method that combines the old method and the added method for the new subtask will be generated to solve a problem under the negative precondition. For example, in Figure 3, the precondition *(?v at ?l1)* is denied to *(not (?v at ?l1))*, so a new task *(drive ?v ?l1' ?l1)* with *(not (same ?l1 ?l1'))* can be added to achieve *(?v at ?l1)*. Then the newly generated method can be applied to a problem with the preconditions *(?v at ?l1')* and *(not (same ?l1 ?l1'))* instead of *(?v at ?l1)*.

However, for those unsatisfied preconditions that cannot be achieved by a new task, the original solution should be modified by using the already existing methods or proved hypothetical methods. Then hypotheses can be proposed. For example, when *(same ?c1 ?c2)* is negated as *(not (same ?c1 ?c2))* for which multiple possible methods can be proposed: one is that a truck is still used to drive the package from *?l1* to *?l2* if there is route connecting *?l1* and *?l2*; another is that the task is re-decomposed into three tasks: first, deliver *?p* to a train station, then a variable of train type is used to deliver *?p* to another train station in *?c2*, then deliver *?p* to *?l2*.

Diversification

Diversification strategy selects a new precondition semantically different from the original preconditions but not conflicting with any of the original predictions; then combines this new precondition to the originals. Semantical difference is defined as that in an ontology, the concept related to a new precondition and the concepts related to original preconditions do not have any common ancestors. This strategy intends to generate methods for those rare situations that seldom happen in example problems and solutions.

For the example in Figure 1, a new subclass can be added to *Thing* such as *Weather* where *Weather* is considered as semantically different from *Package* and *Location*. Two preconditions can be related to *Weather*, such as *(Weather is good)* and *(Weather is snowing)*. Different methods may be preferred under the two weather preconditions, e.g. using a method employing a train may be preferred under the precondition of snow weather.

Generalization

This strategy generalizes a variable from its class to its parent class in an ontology. This strategy should be applied before all the class's siblings in an ontology have been examined. This strategy only chooses one hypothesis that can cover most of the individual methods that can be applied to the children classes. Exclusive preconditions are added to remove any conflictions from the children classes. This strategy makes the new method more general and provides a possible solution for those problems that are not solved by the original theory.

For example, when the hypothetical methods related to *Medium Package* and *Large Package* have been generated by adapting the method in Figure 3 and have been tested, the method shown in Figure 5 can be generalized from *Small Package*, *Medium Package* and *Large Package* to *Package*. Because *Large Truck* can work for all kinds of packages (*Small*, *Medium* and *Large*), this method is chosen as a general method. If *(?v class: Large Truck)* is changed to *(?v class: Medium Truck)*, then a precondition, *(not (?p class: Large Package))*, should be added to exclude the incorrect solution that *Medium Truck* is used to deliver *Large Package*.

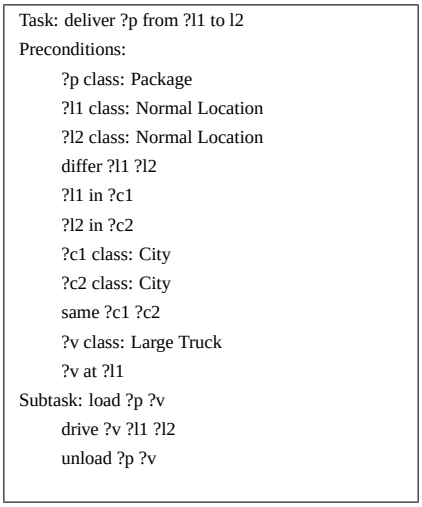

Figure 5: Hypothesis generated by generalization.

Ranking Hypotheses by Heuristics

When multiple hypotheses are generated, they should be evaluated to confirm that they are worthy experimental test operated by a robot or a human. In our approach, the hypotheses are ranked with the following considerations:

- assessing corresponding strength of a strategy that is used to propose a hypothesis;
- evaluating the plausibility of a hypothesis; namely, a hypothetical method has a higher likelihood to be chosen to solve a problem or to succeed when it is applied to a practical problem;
- estimating the novel discoveries that might be produced by a hypothesis.

Based on these considerations, we define the following heuristic function to evaluate a hypothesis:

$$Evaluate(h) = w_s * P_h * \sum(I_h) \quad (2)$$

where, h is a hypothesis, w_s is the weight of the strategy used to propose h, P_h is the plausibility estimated for h based on the underlying computation methods used in the exploration strategies, and I_h represents the estimation of novelties of h's items interesting to a user or to a complete domain theory. The purpose of this function is to balance two factors related to a hypothesis, i.e. the appropriateness of a hypothesis and the interests of domain knowledge discovery or user preference. The appropriateness is encoded in the plausibility of execution success when the hypothesis is applied to a problem or the confidence that the hypothesis is selected from multiple choices to accomplish a task. The discovery of a hypothesis is represented by the summed discovery scores estimated for the items involved in a hypothesis.

For a general HTN domain, the plausibility of a hypothesis can be estimated differently for different exploration strategy together with the background knowledge and the already known planning theory. Our approach makes the estimation by follows:

- For analogical expansion, the computational method as shown in Equation (1) that estimates the similarity of two classes in an ontology can be used to estimate the plausibility of a hypothesis.
- For the strategy of generalization, the plausibility is estimated by the percentage of coverage of the new hypothesis over the methods of the children classes.
- For negation, the confidence of a hypothesis can be evaluated by the success possibilities of the primitive tasks that may estimated from the previous cases.
- Because analogical expansion provides various hypotheses of lower classes that are bases for the strategy of generalization, the strategy of analogical expansion is assigned higher priority.
- For diversification, the plausibility of a hypothesis may be based on background knowledge if related knowledge exists; otherwise, diversification will be assigned the lowest priority.

The general items of discoveries for an HTN planning domain can be categorized into groups with weights indicating preference to complete a domain theory or to satisfy a user's interest. As shown in Figure 6, this kind of estimated weights of discoveries can be defined or modified based on specific domains with the interestingness of specific items domain-dependently defined and estimated.

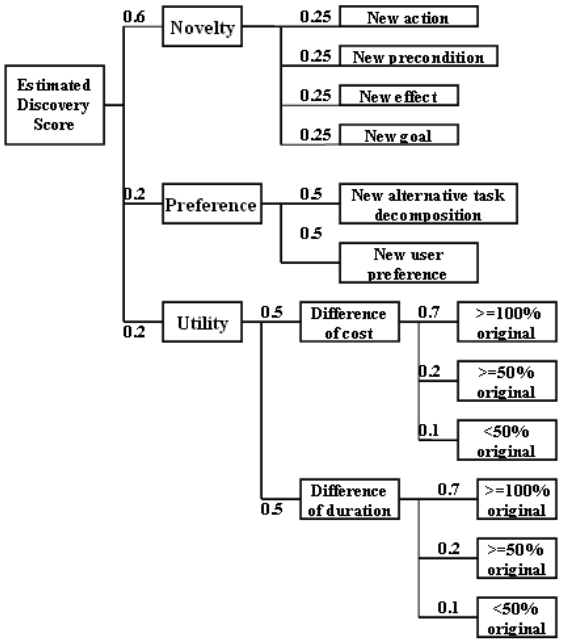

Figure 6: Abstraction of discoveries and their weights.

Plan Execution, Observation and Diagnosis

A hypothetical method generated may be incorrect or incomplete so that a plan solution based on hypothetical models (called a hypothetical plan) may fail when it is executed. Actually, a hypothetical plan provides information about what is expected to happen during an execution. When a prediction is different from what is observed in an execution, it means that an expectation failure happens. Such failures provide critical information for learning and refining a planning theory (Birnbaum et al. 1990; Ram, Narayanan, and Cox 1995), especially for testing and improving a hypothetical model in our approach.

We propose to apply model-based diagnosis technology (Reiter 1987) to diagnose a hypothetical plan execution by observing, comparing and analyzing the differences between predictions and observations of the plan. The incorrectness of hypothetical models can be identified and refined by the diagnosis information.

To test a generated hypothetical HTN task method, the following steps are necessary:

- Design a plan case to test the hypothetical method. A testing plan can be created by instantiating a task method with the instances of that variables. For the example of the method in Figure 4, a hypothetical plan case similar to the case in the left side of Figure 3 can be generated by replacing the variables with instances of the variable classes. For a hypothetical method containing variables of the upper class in an ontology, the variables should be instantiated with each instance of leaf classes; therefore, multiple hypothetical plan cases may be generated to test the model.
- Design observations to monitor during a plan execution. Figure 7 shows how to decide what to be observed for a hypothetical plan case. Here, we assume that a monitor robot or a human is able to decide when an action begins being executed and is able to know when this action is finished. We define that a precondition or an effect that can be monitored is observable.

- Diagnose a plan execution. Based on the difference between observations and predictions, decide if an execution succeeds or fails and give out the reasons of a failure. The diagnosis of one case may be saved in the background knowledge base to guide the future hypothesis generation.

function Observable(hp)
input : an HTN plan $hp = \{T, S_0, A, D\}$, T: task, S_0: initial state, A: action, D: domain knowledge
output : a set of atoms to be observed
$S \longleftarrow \emptyset$;
foreach *action* $a \in A$ **do**
  foreach *precondition* c *of* a **do**
    if c *is observable and* $(c, before(a))$ *not* $\in S$ **then**
      insert $(c, before(a))$ into S;
  endfor
  foreach *effect* e *of* a **do**
    if e *is observable and* $(e, after(a))$ *not* $\in S$ **then**
      insert $(e, after(a))$ into S;
  endfor
endfor
return S;

Figure 7: Designing an observable set of a plan.

Empirical Evaluations

To evaluate our approach, we adopt the variant of the UM Translog domain presented by Xu and Muñoz-Avila (2005) to do experiments for exploring HTN domain by generating hypotheses using the strategies described in the previous section. The domain contains trucks, trains and airplanes to transport packages of various sizes between different sites in different distance ranges (intercity and intracity) as shown in Figure 1. One region contains one or more cities with each city having one or more city-locations that may be transport centers or normal locations that are not transport centers. The transport centers include airports or train stations, while the normal locations serve as the origin or destination of a package. Different kinds of transportation tools are used for deliveries over different distance with different cost. For example, a truck is used for intercity delivery, a train or an airplane is used for intracity transportation. It assumes that any two intercity locations are connected by a truck route, two train stations are connected by a train route, and an airplane route connects two airports.

The initial states are generated with 5 cities each having one airport and one train station, 25 trucks, 20 trains, 20 airplanes and 20 packages. Each of the vehicles is initially located in a random city and randomly categorized into big, medium, and small types. The packages of different types are randomly located at various locations. We randomly generate a set of 150 solvable problems from which 50 problems are randomly selected to consist a seeding set and the left 100 problems become a testing set. For our purpose, JSHOP2 system (Ilghami 2005) is used to simulate solving a problem in the seeding set that requires a domain description, including operators and methods to generate plans. The problems and their corresponding plan solutions (including plans and task decomposition structures) are stored as cases in the seeding set from which HTN-Explorer abstracts a seeding model by replacing an instance object with a variable of its type. The HTN-Explorer will use the seeding model and the ontology to explore a more comprehensive model that will be tested by using JSHOP2 to solve the problems in the testing set.

We conduct evaluation by choosing various numbers of examples from the seeding set. The completeness of a seeding model depends on how much of the domain theory is covered by the selected solution cases. In detail, the experiments are conducted as the following:

1. for N=1 to 50 do steps 2 to 5:
2. Randomly choose N problems with their solutions from the seeding set. Extract an HTN model from each solution and combine the models together to be an initial seeding model.
3. Solve the problems in the testing set using the seeding model.
4. Apply the strategies described in the previous section to the seeding model to generate a more comprehensive model.
5. Use the comprehensive model to solve the problems in the testing set.

Figure 8 shows the results of an experiment in which one circle of hypothesis generation is applied to expand the seeding models that are respectively extracted from the various number of examples in the seeding set. From Figure 8, we can see that the HTN-Explorer can expand an incomplete domain theory efficiently. Figure 9 shows the average number of those strategies that the HTN-Explorer uses to do the experiments as shown in Figure 8. Because the experimental domain is not so complex that the strategy "diversification" is not applied.

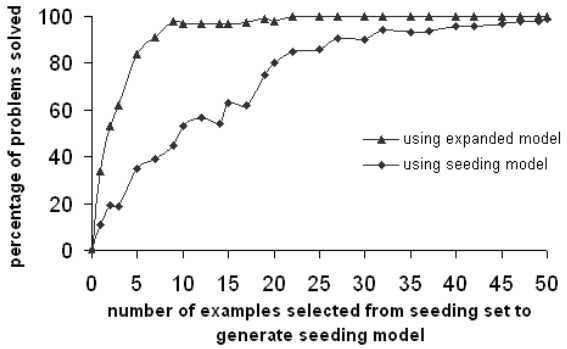

Figure 8: Experimental results of seeding models and expanded models after one circle of hypothesis generation.

Related Work

Automated HTN planning requires that a domain theory (descriptions of actions and methods) be present to a plan-

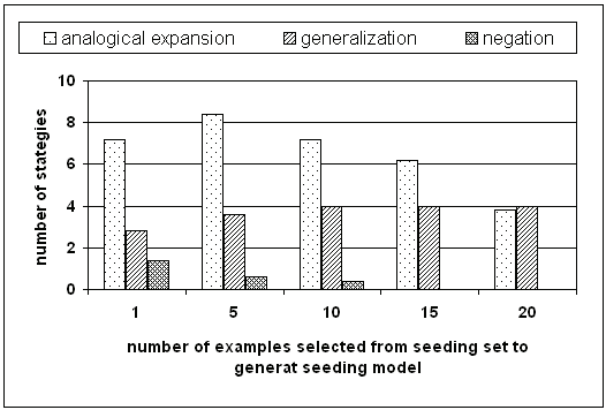

Figure 9: Strategies used to explore seeding models in Figure 8.

ner. Besides building up tools to facilitate effective planning domain acquisition and validation (e.g. GIPO (McCluskey, Liu, and Simpson 2003) and itSIMPLE (Vaquero et al. 2009)), researchers have been interested in pursuing (semi)autonomous programs to generate a domain theory for those complicated empirical domains.

CHEF and PRODIGY/ANALOGY (Veloso and Carbonell 1993) are two representative systems that extend planning domain knowledge with the analogical knowledge and case-based reasoning. While CHEF employs domain-dependent reasoning knowledge, PRODIGY/ANALOGY develops completely domain-independent analogical reasoning mechanisms and uses cases as search control knowledge. PRODIGY/OBSERVER (Wang 1996) automatically acquires and refines the preconditions and effects of operators by observing expert solution traces. PRODIGY/EXPO (Carbonell and Gil 1990) refines incomplete operator models by proposing experiments to test the explanations for those observed divergences. However, none of these systems does task model learning.

Instead of requiring a large numbers of training cases as those case-based reasoning systems, some authors have proposed an explanation-based learning approach that can learn from a single training example with the aid of the domain knowledge (DeJong and Mooney 1986). For example, GRASPHER (Bennett and Dejong 1996) implements a permissive planning approach to acquire and refine generalized plan schema through explanation-based learning.

To learn or improve the hierarchical structures relating tasks and subtasks (task models), one approach learns preconditions of HTN methods from HTN plan solution examples, e.g. CaMeL(++) (Ilghami et al. 2005) and DInCaD (Xu and Muñoz-Avila 2005). The other approach elicits the hierarchical structures of tasks from a collection of STRIPS action plans and hard-coded hierarchical annotation, e.g. HTN-MAKER (Hogg, Muñoz-Avila, and Kuter 2008). The generalization strategy proposed in this paper is similar to the approach described in DInCaD; however, the other exploration strategies presented in our work make our approach go further in expanding planning domain knowledge. The main difference between our approach and these previous case-based systems is that the previous works can only update their knowledge bases of HTN methods when they are provided with new problem solution cases; thus, they cannot produce methods that are not captured in the plan examples.

Another approach to exploiting hierarchies in planning is abstraction, such as ABSTRIPS (Sacerdoti 1974) and ALPINE (Knoblock 1990). These systems use both a collection of operators and an abstraction model that benefits the search process. Newton et al. (2008) learn control knowledge not captured by examples with genetic approach.

In summary, different from the previous approaches, HTN-Explorer integrates various strategies with an aim at self-directed exploring a bigger search space that example data does not provide. HTN-Explorer provides a flexible framework to integrate general domain-independent HTN exploration strategies and domain specific background knowledge represented in an ontology. The strategies HTN-Explorer implements to expand a domain space are not totally example-dependent. In addition, HTN-Explorer implements a heuristic function for evaluation of hypotheses. Generally, HTN-Explorer is like a knowledge discovery system, such as AM (Lenat 1982) and HAMB (Livingston, Rosenberg, and Buchana 2003). While AM focuses on mathematics domain and HAMB concentrates on chemistry discovery, our work focuses on exploring an incomplete HTN model.

Conclusion and Future Work

In this paper, we propose a framework that present hypotheses to explore an incomplete HTN planning domain. We present multiple exploration strategies and a heuristic function to estimate the value that a hypothesis deserves experiments to provide new theory for a domain. Finally, we demonstrate an empirical evaluation to test the effectiveness of our approach with the UM Translog domain.

For future work, more useful exploration strategies will be added to HTN-Explorer. In this paper, we have not considered the utilities of a plan execution, such as cost and duration. In real world, these properties are also important for hypothetical model generation. We will add utility consideration into a hypothesis evaluation. In addition, in this paper, we have only taken the advantage of the hierarchical structure of an ontology. The future work will study how to use the detailed properties of concepts in an ontology to propose and evaluate hypothetical models.

In our opinion, a planning domain model (HTN or non-HTN) can be improved by our approach. For example, an operator model can be expanded by the hypothesis strategies we presented. In addition, our approach can be applied to discover users' preference or to discover interesting knowledge for some real world domains whose background knowledge can be represented in HTN formalism. In the future, we will apply our approach to more planning domains to test its effectiveness.

References

Andrews, S.; Kettler, B.; Erol, K.; and Hendler, J. 1995. Um translog: A planning domain for the development and benchmarking of planning systems. *Technical Report, Dept. of CS, Univ. of Maryland at College Park*.

Bennett, S. W., and Dejong, G. F. 1996. Real-world robotics: Learning to plan for robust execution. *Machine Learning* 23(2-3):121–161.

Birnbaum, L.; Collins, G.; Freed, M.; and Krulwich, B. 1990. Model-based diagnosis of planning failures. In *Proceedings of AAAI'90*, 318–323.

Carbonell, Y. G., and Gil, Y. 1990. Learning by experimentation: The operator refinement method. *Machine Learning* 3:191–213.

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

DeJong, G. E., and Mooney, R. J. 1986. Explanation-based learning: An alternative view. *Machine Learning* 1(2):145–176.

Erol, K.; Nau, D.; and Hendler, J. 1994. Htn planning: Complexity and expressivity. In *Proceedings of AAAI'94*, 1123–1128.

Estlin, T.; Castano, R.; Anderson, B.; Gaines, D.; Fisher, F.; and Judd, M. 2003. Learning and planning for mars rover science. In *Proceedings of IJCAI'03*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kauffmann.

Hogg, C.; Muñoz-Avila, H.; and Kuter, U. 2008. Htn-maker: Learning htns with minimal additional knowledge engineering required. In *Proceedings of AAAI'08*.

Ilghami, O.; Muñoz-Avila, H.; Nau, D. S.; and Aha, D. W. 2005. Learning approximate preconditions for methods in hierarchical plans. In *Proceedings of ICML'05*, 337–344.

Ilghami, O. 2005. Documentation for jshop2. Technical Report CS-TR-4694, University of Maryland, Department of Computer Science.

Knoblock, C. 1990. Learning abstraction hierarchies for problem solving. In *Proceedings of AAAI'08*, 923–928.

Lenat, D. 1982. Am: Discovery in mathematics as heuristic search. *Knowledge-Based Systems in Artificial Intelligence* 3–225.

Livingston, G.; Rosenberg, J.; and Buchana, B. 2003. An agenda- and justification-based framework for discovery systems. *Knowledge and Information Systems* 5:133–161.

McCluskey, T. L.; Liu, D.; and Simpson, R. 2003. Gipo ii: Htn planning in a tool-supported knowledge engineering environment. In *Proceedings of ICAPS'03*.

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

Nau, D. S. 2007. Current trends in automated planning. *AI Magazine* 28(4):43–58.

Newton, M. A. H.; Levine, J.; Fox, M.; and Long, D. 2008. Learning macros that are not captured by given example plans. In *Supplementary Online Proceedings for Poster Papers at ICAPS'08*.

Ram, A.; Narayanan, S.; and Cox, M. T. 1995.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial Intelligence* 32(1):57–96.

Sacerdoti, E. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5(2):115–135.

Sacerdoti, E. 1975. The nonlinear nature of plans. In *Proceedings of IJCAI'75*, 206–214.

Smith, S. J. J.; Nau, D. S.; and Throop, T. 1998. Computer bridge: A big win for ai planning. *AI Magazine* 19(2):93–105.

Vaquero, T. S.; Silva, J.; Ferreira, M.; Tonidandel, F.; and Beck, J. 2009. itsimple3:0: From uml requirements and petri net-based analysis to pddl representation in the process of modeling plans for real applications. In *Proceeding of ICAPS 2009 Workshop on Knowledge Engineering for Planning and Scheduling*.

Veloso, M. M., and Carbonell, J. G. 1993. Derivational analogy in prodigy: Automating case acquisition, storage, and utilization. *Machine Learning* 10:249–278.

Wang, X. 1996. A mulitstrategy learning system for planning operator acquisition. In *Proceedings of the Third International Workshop on Multistrategy Learning*.

Wilkins, D. 1985. Recovering from execution errors in sipe. *Computational Intelligence* 1:33–45.

Xu, K., and Muñoz-Avila, H. 2005. A domain-independent system for case-based task decomposition without domain theories. In *Proceedings of AAAI'05*, 234–240.

Zimmerman, T., and Kambhampati, S. 2003. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine* 24(2):73–96.