TOWARDS INDUCING HIERARCHICAL TASK NETWORK DOMAIN MODELS FOR AI PLANNING FROM EXAMPLES

N. E. Richardson
University of Huddersfield, Queensgate, Huddersfield HD1 3DH, UK

ABSTRACT

Domain modelling for AI Planning aims to form a database of facts about the 'world' being modelled. This can be a complex process especially if there is a large number of objects or actions or both to be modelled. This task can be facilitated by tools which induce operators or methods from examples. Further, large and complex domains are more easily constructed if domain languages are used which allow for hierarchical decomposition of domain components. Examples of such a decomposition are ***class hierarchies*** *and* ***method hierarchies****. This paper describes ongoing work which aims to produce algorithms which learn effective hierarchical decompositions from examples.*

Keywords domain model, operators, methods, GIPO.

1 INTRODUCTION

Domain modelling is a complex, error prone process, especially when the model is complex. Within this model the most difficult parts to capture are the allowed actions within the world being modelled. Actions are modelled as ***operators*** and more complex sequences of actions are modelled as ***methods.*** Capturing the dynamics and a concept of the allowed behaviours using operator or method structures lies at the heart of constructing planning domains. One way to facilitate the process is to use tools which induce operators or methods using task solutions as training examples. In our previous work we have shown how `flat' (non-hierarchical) domain operators can be induced from examples.

McCluskey et al (2002) show that operators can be induced using ***opmaker*** which has been embedded interactively in ***GIPO***, Simpson et al (2001) and Simpson (2005). ***GIPO*** (Graphical Interface for Planning with Objects) is a tool which aids domain construction, offering editors, validation tools, a graphical life-history editor and planning tools. Output from ***GIPO*** is the completed and validated domain being modelled in a variant of ***GIPO***'s internal language ***OCL,*** Liu et al (1999). ***GIPO*** will also produce output in the universally accepted planning language, ***PDDL***.

Large and complex domains are more easily constructed if domain languages are used which allow for hierarchical decomposition of domain components. This makes for a richer language which more closely captures the real world situations. Methods composed of hierarchical task networks (HTN) make better sense of these worlds but are, however, difficult to construct. We are working on an extension of the induction process whereby operators are combined into task networks. To illustrate the techniques we are using we have created a hierarchical version of a domain model familiar to the planning community and known as the ***briefcase*** domain. Below we briefly describe this work towards creating procedures which input training sequences and a partial model containing object and class information, and outputs a HTN domain model.

2 HIERARCHICAL DOMAINS

A planning domain is simply a knowledge base containing facts about the world being modelled. In classical planning the world is closed so that all known facts are contained in the knowledge base. The world consists of objects, and activities happen as the result of the application of an operator. Operators may be simple (primitive) and complete a single action, or they may complete a series of actions (methods) due to the application of all their specified actions. Objects exist in states (for example a book may be `on the table' or `held' and the action `pick_up' changes its state) so an operator describes what state changes occur to the objects involved. It follows that an object undergoes a series of state changes if a method is applied to it.

To illustrate our work we refer to our new version of the briefcase world containing a simple structural hierarchy of object "sorts" shown in figure 1. The tree shows the hierarchical sort structure with predicates attached at appropriate levels. For example inheritance in the sort tree means that the state

at_carrier applies not only to carrier but to any other sort below it on the tree. The converse does not work so that goes_in applies to box, lunch_box and pencil_box only, and not to carrier or bag.

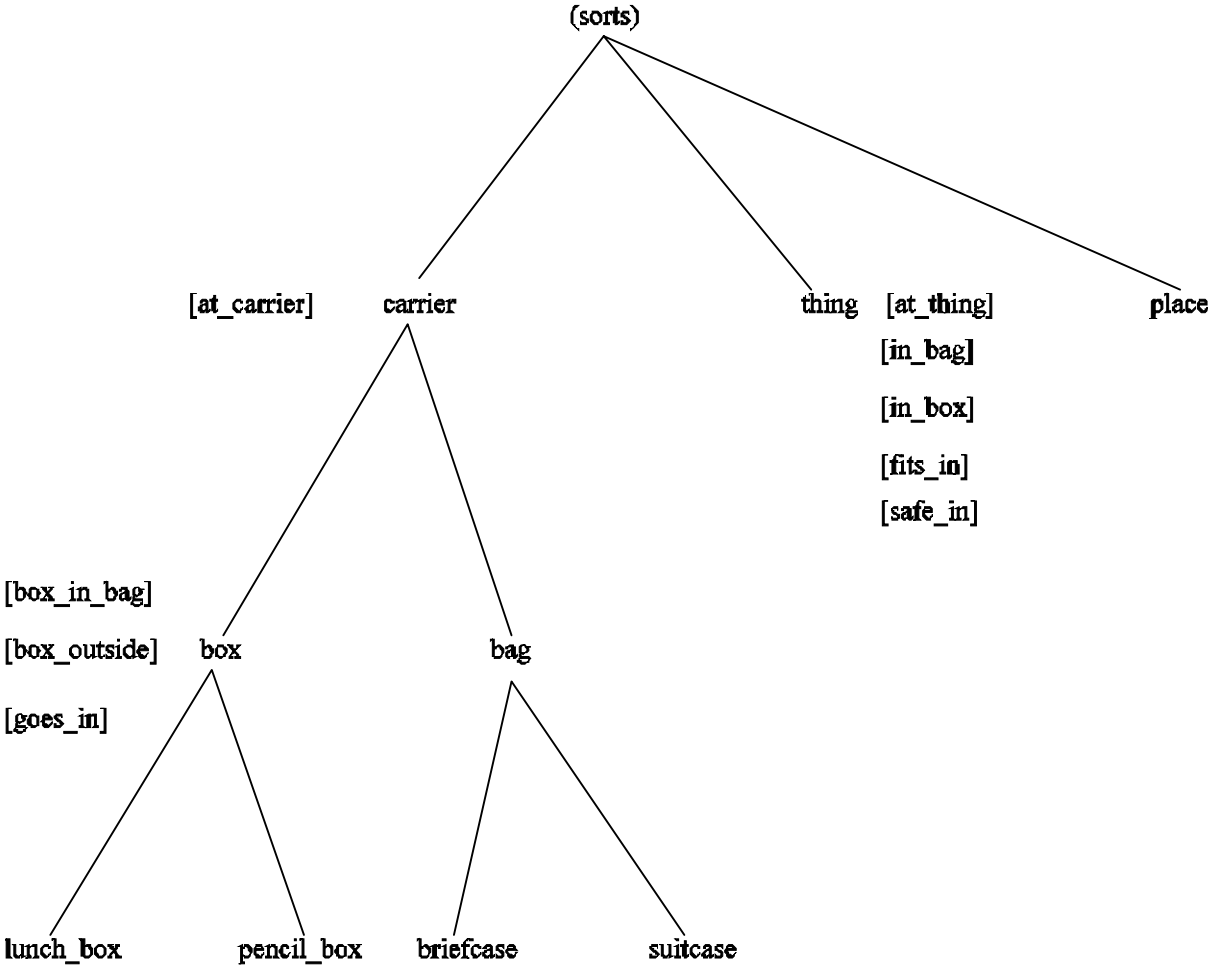

Figure 1: The Sort-Tree Showing the Levels at which Predicates Apply

In the ***OCL*** language planning domains may be hierarchical in two ways. The language structures the objects to be members of certain types called `sorts'. For example in the hierarchical briefcase domain (HBC)

```
sorts(carrier,[bag,box]).
sorts(bag,[briefcase,suitcase]).
objects(briefcase,[bc1]).
objects(suitcase,[sc1]).
```

describes how bag (and box) are of sort `carrier', whilst briefcase is of sort `bag' and `bc1' is a specific object of sort briefcase. The second example of the hierarchical nature of domains involves the methods. Methods are constructed because the sequence of actions they perform need to be packaged together for efficiency and/or effectiveness, and hence they encapsulate domain heuristics. They can be thought of as `mini-plans' where a plan is a sequence of actions to achieve the state changes from a specified initial state to some predetermined goal state. Methods are structured into hierarchies so that some methods decompose into others or decompose into both other methods and

primitives in order to complete their task. This structure in complex domains can be quite extensive and it can be difficult to see the interlacing of tasks.

3 WORK IN PROGRESS

Using ***GIPO*** we constructed the new briefcase domain version complete including operators, methods and domain tasks. The latter consist of some achievable sets of initial states and goal conditions and serve as planning challenges – they set a task to the planner to find a sequence of operators that will achieve the goal state given the initial state. The newly constructed complete domain was then used as a benchmark for the operators we were going to induce.
Using partial domain models we have been able to replicate the ***GIPO***-constructed operators and methods by induction as follows. Example files containing the partial domain (including an object class hierarchy) but excluding all the operators, methods and tasks are compiled. These files each contain a solution to a planning task in the form of a named operator sequence, initial states for the objects involved and numbered example material indicating the states after the application of each operator. The induction algorithm outputs a set of instantiated operators and an HTN method induced from the sequence. In each case only those operators required for the methods we were replicating were induced from each file. An example induced operator put_box_in_bag, where state transitions are written as pre-action state → post-action state, is as follows.

```
operator(put_box_in_bag(Bag,Place,Box),
   %prevail
   [se(bag,Bag,[at_carrier(Bag,Place)])],
   %necessary
   [sc(box,Box,[box_outside(Box), at_carrier(Box,Place)] =>
   [box_in_bag(Box,Bag),at_carrier(Box,Place),goes_in(Box,Bag)])],
   %conditional
   [sc(thing,Thing,[in_box(Thing,Box),at_thing(Thing,Place)] =>
   [in_box(Thing,Box),at_thing(Thing,Place),safe_in(Thing,Box)])]).
```

Here the operator header lists the sorts of objects involved in the action and the prevail transition states that the bag remains at the same place. The necessary transition states that the box changes state from being outside the bag at a place to being inside the bag at the same place. Finally the conditional transition states that if a thing is in the box then it also undergoes a state change - in this case it is still in the box but the box is now in the bag.
A simple method operator induced is shown below. The task network is composed of two induced operators put_in_box and put_box_in_bag.

```
method(pack_lunch(Sandwiches,Place,Lunch_box,Bag),
   % pre-condition
   [],
   % Index Transitions
   [sc(thing,Sandwiches,[outside(Sandwiches), at_thing(Sandwiches,Place)]=>
                  [in_box(Sandwiches,Lunch_box),at_thing(Sandwiches,Place)]),
   sc(lunch_box,Lunch_box,[box_outside(Lunch_box),
         at_carrier(Lunch_box,Place)]=>
       [box_in_bag(Lunch_box,Bag), at_carrier(Lunch_box,Place)])],
   % Static
   [safe_in(Thing,Lunch_box),
    goes_in(Lunch_box,Bag)],
```

```
% Temporal Constraints
[before(1,2)],
% Decomposition
[put_in_box(Box,Place,Thing),
 put_box_in_bag(Bag,Place,Box)]).
```

This format allows for any preconditions to be listed and the main transitions for the lunch_box and sandwiches are listed. The decomposition names the two operators of which this method is composed and the temporal constraints name the order in which they must be applied.
The new operators and methods have been compared to the hand constructed set (using ***GIPO***). Our initial results show that the induced sets are accurate: when used with ***GIPO***'s planner and stepper tools we were able to complete all the tasks previously declared for the domain. Preliminary tests (with GIPO's planner HyHTN) show that plans formed using just induced operators run faster than those formed using both induced methods and operators but this may be because of the simplicity of the domains used. Further tests using more complex domains such as the Tyre World indicate that as the tasks become more complex, inducing methods as well as operators improves planning efficiency. Recently we have been able to demonstrate that planning is enhanced by the use of induced methods. Experimentation with the tyre domain has shown that a correct sequence of eleven operators specified in three methods found a plan solution in less than a second whilst without the use of methods the planner had failed to find a solution after 4 days running on a Sun `SunBlade 100'. We aim to show that as methods are learned and new methods are induced that utilise them, we can build induced method hierarchies for more complex real world situations.

4 RELATED WORK

We know of no systems that learn effective hierarchical task network (HTN) operator sets by inducing both primitive operators and methods. The argument for more complex, structured operators to be used to model the difficulties faced when in real world situations is well put by Levine and DeJong (2006). Their solution to the problem is similar to ours and they introduce a system of automatically constructing planning operators. The difference is that they shield the planner from all but the necessary elements which are visible to the planner.
Garland, Ryall and Rich (2001) show, in their Collagen system, that learning task models can be achieved by training examples and support from a domain expert. Their work is similar to our approach in the following ways:

- their `task models' are similar to our HTN methods
- they show a complete recipe to achieve some task
- they show orderings of the steps to achieve the task
- they are developing a graphical user interface to aid construction
- the orderings of the steps contain primitive and non-primitive stages
- they list constraints that apply to the various steps
- user/expert guidance is required for the detail.

A more recent system that learns operators from examples is ARMS, Wu et al (2005). This system learns operator specifications without the need for user intervention or a partial domain specification. However, it requires many training examples containing valid solution sequences, and presently it is only capable of inducing "flat" operators.

REFERENCES

GARLAND A, RYALL K and RICH C (2001). *Learning Hierarchical Task Models by Defining and Refining Examples.* Proceedings of the First International Conference on Knowledge Capture.

LEVINE G and DeJONG G (2006). *Explanation-based Acquisition of Planning Operators.* Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, pages 152-161.

LIU D and McCLUSKEY T L (2000). *The OCL Language Manual, Version 1.2.* Technical report, Department of Computing and Mathematical Sciences, University of Huddersfield.

McCLUSKEY T L, RICHARDSON N E and SIMPSON R M (2002), *An Interactive Method for Inducing Operator Descriptions.* Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems.

SIMPSON R M, McCLUSKEY T L, ZHAO W, AYLETT R S, and DONIAT C (2001). *GIPO: An Integrated Graphical Tool to Support Knowledge Engineering in AI Planning*. Proceedings of the Sixth European Conference on Planning.

SIMPSON R M (2005). *GIPO Graphical Interface for Planning with Objects.* Proceedings of the National Conference for Knowledge Engineering in Planning and Scheduling.

WU K, YANG Q and JIANG Y (2005). *Action-relation Modelling System for Learning Acquisition Models.* Proceedings of the First International Competition on Knowledge Engineering for AI Planning. Monterey, California, USA.