A STUDY OF SYNTHESIZING ARTIFICIAL INTELLIGENCE (AI) PLANNING DOMAIN MODELS BY USING OBJECT CONSTRAINTS

M. M. S. Shah, T. L. McCluskey And M. M. West
University of Huddersfield, Queensgate, Huddersfield HD1 3DH, UK

ABSTRACT

This paper concerns the area of automated acquisition of planning domain models from one or more examples of plans within the domain under study. It assumes that an adequate domain model for a domain can be composed of objects arranged in collections called object sorts. Recently, two systems have had success in using this underlying assumption: the Opmaker2 system (McCluskey et al. 2009), and the LOCM system (Cresswell, McCluskey, and West 2009). The former requires only one solution plan as input, as long as it contains at least one instance of each operator schema to be synthesized. It does require a partial domain model as well as the example plan, and the initial and goal states of the plan. In contrast LOCM requires no background information, but requires many instances of plans before it can synthesize domain models. Our aim is to build on these systems, and establish an experimental and theoretical basis for using object - centred assumptions to underlie the automated acquisition of planning domain models.

Keywords Knowledge acquisition, object-centred model, domain model

1 INTRODUCTION

Automated planning is an important feature for any intelligent system to increase its autonomy and flexibility through the construction of sequences of actions to achieve its goal. Planning techniques have been applied in a variety of tasks including robotics, process planning, fire fighting, web-based information gathering, autonomous agents, and NASA's Mars Rover. Artificial Intelligence (AI) planning is a booming research area for last two decades, but the significant technology has only begun to extent up to problems of rational size. Accordingly, research interest in the planning technology is growing rapidly, creating many opportunities for industrial and commercial applications.

This research work concerns the area of automated acquisition of full or partial domain models from one or more examples of plans within the domain under study. One motivation is that the knowledge engineering of such domain models by hand into languages such as PDDL is inefficient and laborious. Another is to help planning agents become more autonomous. Agents that have planning capabilities may need the ability to acquire and refine domain models, if they encounter new domains.

Our work assumes that an adequate domain model for a domain can be composed of objects arranged in collections called object sorts. We use the assumptions of Simpson et al's object-centric view of domain models (Simpson, Kitchin, and McCluskey 2007). Here a planning domain model consists of sorts of object instances, where each object behaves in the same way as any other object in its sort. Sorts have a defined set of states that their objects can occupy, and an object's state may change (called a state transition) as a result of a domain action's execution.

Recently, two systems have had success using this underlying assumption: the Opmaker2 system (McCluskey et al. 2009), and the LOCM system (Cresswell, McCluskey, and West 2009). The former requires only one solution plan as input, as long as it contains at least one instance of each operator schema to be synthesized. It does require much background knowledge, however, including a partial domain model, as well as the training plan itself, and the initial and goal states of the plan. In contrast LOCM requires no background information, but requires many training plans before it can synthesize domain models. We aim to build on these systems, and establish an experimental and theoretical basis for using object - centred assumptions to underlie the automated acquisition of planning domain models.

2 RESEARCH PROBLEM

In AI planning, domain modelling is a complicated process especially if there are a large number of objects or actions or both are to be modelled. Capturing dynamics and behaviour using operator structures lies at the heart of constructing planning domains.

One can classify work by considering what inputs the systems require, and what their outputs are, as given in the list below. Let us assume a system ”observes” training plans containing action instances, and tries to synthesize operator schema representing groups of these action instances. The features of the systems may include the following.

- **Observations(s):** Are many sequences of training plans required or just one? Are the plans assumed to be correct? How are they represented e.g. as an action name followed by a list of parameters representing objects affected?
- **Training help:** As well as the observed plan, is there other data supplied? e.g. a system might be supplied with the initial and goal state that the plan solves, as well as partial or full intermediate state information.
- **Constraints:** is there a background set of constraints assumed about the world which help in the synthesis stage? For example, in the object-centric approach we assume that objects are affected by actions, and an object is input to an action in the state that it leaves the previous action.
- **Partial Model:** is there a pre-defined specified language of predicates or fluents? e.g. most systems assume that the user has specified a set of fluents in which the world is described, thus constraining the system's range of applications
- **Algorithm:** this refers to the characteristics of the algorithm used to synthesize operators: does it use deduction and/or induction, or hill climbing or other search techniques?
- **Output:** this is normally a set of operator schema or methods, but could also include a set of heuristics to be used to speed up planning. The expressiveness of the output schema language is important in determining where a system is capable of adequately synthesizing a domain model. One way of characterising this is to use the levels in PDDL as a form of measure.

3 DOMAIN MODEL

To motivate the work on synthesizing domain models from an object perspective, we use an example from computer games, called the “Ring World”. The idea is, for each group of characters in the game, we need to create a domain model specifying the actions that they can take. After this, a planning engine can then be used to generate plans and create the illusion of goal directed behaviour. The game was designed as follows: there is one overall ‘database’ which can completely describe any state of the game. Different sets of characters have different sets of capabilities: one is a set of knights that attempt to bring about a set of goals (eg acquire the Ring, kill orcs). Another is a set of orcs, with goals such as blocking doors, disabling knights etc. Each set of characters can use deliberative planning to try to achieve their goals, execute their plans and replan when their plans are no longer executable. The player character within the game could, for example, play the role of a ‘wizard’ that observed one force’s plans and attempted to help achieve them.
A step towards implementing the game might be to construct a planning domain model with operator schema representing the capabilities of the characters, and the effects of their actions. Sets of operator schema, representing the views of the overall domain model, have to be generated for this, one for each set of characters. These views would constitute distinct operator sets, whereas the object structures would be common to all and constitute the central database. An example snapshot (or state) of the “Ring World” is illustrated in Figure 1. For example, from the point of view of the force controlling the knights:

- knights can move between rooms and passages. Movement will take place via propositional steps such as
  “move-into-room” and “move-to-passage”;
- passages can be blocked by orcs or locked doors;
- locked doors need to be opened by the correct key;
- knights can acquire keys from trolls if they are near them and offer them some treasure in exchange;
- orcs that are sleeping can be killed by a knight if it is near them and has a weapon (however the weapon is then spent);
- orcs that are awake are similar to sleeping orcs except the knight needs to pick up protection first (a shield or a spell) before fighting them;
- hidden treasure can found by a knight if it is in the same room as the treasure.

For example, consider the state in Figure 1. Assume that a goal to be achieved is for the knight “Aragorn” to acquire the ring. Then the planner would have to generate a plan for him to move through rooms and passages to acquire a key to unlock one of the passages leading to r2 (the only route to the ring in r1); and to acquire some treasure so that he can trade it for protection, so that he will be able to overcome the orc guarding room r1, when he eventually arrived there.

Use of Object Centred Constraints in Determining Object Transitions

We will use the Ring World example to illustrate the idea of how object constraints can help with the domain model synthesis process. In this case, we would like to use constraints to determine the complete set of transitions of individual objects as they are affected by actions in a training plan. We assume that a correct training plan is input, with initial and goal state as training help. Additionally, assume that we have partial domain model that includes only a propositional encoding of the Ring World, with the treasure object necklace being described as in one of 4 states: hidden, held by knight, taken by troll, offered for barter.
Assume a training plan of

"Find necklace, Offer necklace, Exchange necklace"

The initial state is "hidden", and the final state is taken by troll. As shown in Figure 2, there are 7 possible paths through the object state space, using the assumption that the object changes state at each step. In this case it would not be possible to trace a deterministic path through the space of propositions, from the training sequence, as the constraints on initial and goal states are insufficient.

If we introduce a relational, object representation, then this introduces extra constraints. Assume that any piece of object treasure (?t) can occupy one of four abstract states, where it might be related to a ring (?r), a knight (?k), or a troll (?m), as follows:

hidden(?t,?r)
holding(?t,?k)
taken(?t,?m)
offered(?t,?k,?m)

Returning to the example with the new representation, assume the initial state is "hidden(necklace,r7)", the final is "taken(necklace,troll1)", and we use the training sequence from the example above. We reproduce this in the syntax used by the Opmaker system (McCluskey et al. 2009), below (where the character '@' is used to distinguish an object that does not change state as a result of an action):

find_treasure(@knight1,@r7,necklace),
move_to_passage(knight1,p67,r7),
present(@george,@p67,@troll1,necklace),
exchange(necklace,key1,@george,@p67)

We can now trace a unique path through the state space of the necklace, using object constraints, from the initial state
"hidden(necklace,r7)". The first four possible transitions are:

hidden(necklace,r7)=> holding(necklace,knight1)
hidden(necklace,r7)=> offered(neckalce,knight1,?troll)
hidden(necklace,r7)=> taken(necklace,?troll)
hidden(necklace,r7)=> hidden(necklace,?r)

Given that the first step in this sequence does not refer to a troll, or two different rooms, only the first of the four possible forms of transition below would be possible, as it is the only one which leads to an instantiated transition. The last transition would only be possible if two distinct rooms had been specified in the sequence step. The transitions effected by move to passage do not affect the necklace, so this step can be ignored. The next step leads to the following transitions:

holding(necklace,knight1)=> holding(necklace,?knight)
holding(necklace,knight1)=> offered(neckalce,knight1,troll1)
holding(necklace,knight1)=> taken(necklace,troll1)
holding(necklace,knight1)=> hidden(necklace,?room)

Hence the second and third transitions are possible. The first is ruled out as there is no new knight object specified to instantiate "?knight", and in the last there is no value specified for "?room". Keeping both these options open, we can examine the last transition, which must change the necklace into state "taken(necklace,troll1)". This constraint eliminate the third option in the step above, leaving transition

offered(neckalce,knight1,troll1) => taken(necklace,troll1)

as the only option available. This unique path of transitions through the object state space of sort treasure is illustrated in Figure 3.

4 PREVIOUS WORKS

There has been a great number of related works in this area under the assumption that the output is a literal-based *STRIPS* like domain model. In this work, it is generally the case that predicates or fluents are predefined, and the aim of the system is to use example plans, either incrementally or all at once, to induce or deduce operator schema in terms of these fluents. We will mention several recent systems.
Opmaker2 (McCluskey et al. 2009) is an extended version of Opmaker (McClusky, Richardson and Simpson 2002) which is embedded with GIPO (Simpson et al. 2001) and (Simpson 2005). In Opmaker authors showed how 'flat' domain actions can be induced from examples. Opmaker2 is introduced in GIPO so that it

can induce action from training sequences and its static object model alone, without any intermediate state information and without many examples. Eventually Opmaker2 reduces the burden of knowledge engineering, so that a program (agent) can perform knowledge acquisition rather than acquiring through human-driven process supported by a tool like GIPO. Opmaker2 requires only one solution plan as input, as long as it contains at least one instance of each operator schema to be synthesized. It requires more background knowledge a partial domain model, Training plan, Initial and goal states.
In the ARMS system (Wu, Yang, and Jiang 2005), a domain model is output in the form of STRIPS-type operator schema, with the input being multiple examples of training plans. ARMS does not require intermediate state information, but requires background knowledge such as types, relations, initial and goal states, and relies on many training plans containing valid solution sequences.
In the SLAF approach (Shahaf and Amir 2006) expressive operator schema can be output (at least to ADL level), but SLAF requires as input specifications of fluents, as well as partial observations of intermediate states between action executions. Learning expressive theories from examples is a central goal in the Inductive Logic Programming (ILP) community, and some systems build on this considerable body of work. For example, in his thesis (Benson 1996), Benson describes an ILP method for learning very expressive operator schema, using multiple examples, in an incremental fashion.
LOCM system (Cresswell, McCluskey, and West 2009) can induce action schema without being provided with any information about predicates or initial goal or intermediate state descriptions, for example: action sequence. LOCM acts from example planes where each plan is a sound sequence of action and the actions change the state of objects where they should before execution. Each plan in LOCM is a sequence of actions and planning traces are input into LOCM as an ordered set of action instances, where each action instances is identified by name plus the object instances that are affected or are necessarily present but not affected by action execution.
There is also a body of research aimed at learning hierarchical domain models, mainly in the “HTN” variety. Practical planning domains are based on ‘hierarchical task network’ decomposition, and it could be claimed that these systems learn heuristics encapsulated within an HTN method. HTNs can be very difficult to construct manually and authors have worked in producing these using methods from machine learning. In (Nejati, Langley, and Konik 2006) the authors describe how they induce teleoreactive logic programs from expert traces. The teleoreactive programs index methods by the goals they achieve. They use methods derived from explanation based learning to chain backwards from the end result of the sample trace. Theoretical work on HTN planning is presented in (Ilghami et al. 2005). This paper introduces a formalism whereby situations are modelled where general information is available of tasks and subtasks, together with some plan traces. In the early work all information about methods was required except for the preconditions. This limitation is overcome in later work by the same group (Ilghami, Nau, and Munoz-Avila 2006) a new algorithm ‘HDL’ (HTN Domain Learner) is presented which learns HTN domain descriptions from plan traces. Between 70 and 200 plan traces are required to induce the descriptions. HTN-MAKER is presented in (Hogg and Munoz- Avila 2007). This receives as input a STRIPS domain model, a collection of STRIPS plans and task definitions and produces an HTN domain model. The experimental hypothesis is that after a few problems have been analyzed an HTN domain model will be ultimately obtained able to solve most solvable problems. A version of the logistics-transportation domain is chosen for the experiment and good results are obtained. However these good results are not replicated for the blocks-world domain. One problem is the large number of methods which have to be learned, where one method might include another.

5 CONCLUSIONS

The aim of this paper is to explore the scope and potential of using object-centred constraints to assist the acquisition of planning domain models from examples. We plan to build on the recent work embodied in the objectcentred systems of Opmaker2 (McCluskey et al. 2009), and LOCM (Cresswell, McCluskey, and West 2009); whereas they have produced promising experimental results, there is a need to analyze them theoretically, to find the limits of their output domain models, and the conditions under which they are successful. In this paper we have reviewed some of the large amount of related work, and introduced a classification for systems that learn planning operator schema. We introduced an example domain to illustrate the kind of constraints that an object centred approach can bring, and stepped through an example sequence showing how a unique object transition path can be determined.

REFERENCES

Benson, S. S. 1996. Learning Action Models for Reactive Autonomous Agents. Ph.D. Dissertation, Dept of Computer Science, Stanford University.

Cresswell, S. N.; McCluskey, T. L.; andWest, M. M. 2009. Acquisition of Object-Centred Domain Models from Planning Examples. In Proceedings of ICAPS 2009.

Hogg, C., and Munoz-Avila, H. 2007. Learning Hierarchical Task Networks from Plan Traces. In Proceedings of the ICAPS'07 Workshop on Artificial Intelligence Planning and Learning.

Ilghami, O.; Nau, D. S.; Muoz-Avila, H.; and Aha, D. W. 2005. Learning preconditions for planning from plan traces and HTN structure. Computational Intelligence 21(4):388–143.

Ilghami, O.; Nau, D. S.; and Munoz-Avila, H. 2006. Learning to do htn planning. In Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, 390 – 393.

McCluskey, T.; Cresswell, S.; Richardson, N.; and West, M. M. 2009. Automated acquisition of action knowledge. In International Conference on Agents and Artificial Intelligence (ICAART), 93–100.

Nejati, N.; Langley, P.; and Konik, T. 2006. Learning hierarchical task networks by observation. In ICML '06: Proceedings of the 23rd international conference on Machine learning, 665–672. New York, NY, USA: ACM Press.

Shahaf, D., and Amir, E. 2006. Learning partially observable action schemas. In AAAI. AAAI Press.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007. Planning Domain Definition Using GIPO. Journal of Knowledge Engineering 1.

Wu, K.; Yang, Q.; and Jiang, Y. 2005. ARMS: Action relation modelling system for learning acquisition models. In Proceedings of the First International Competition on Knowledge Engineering for AI Planning.

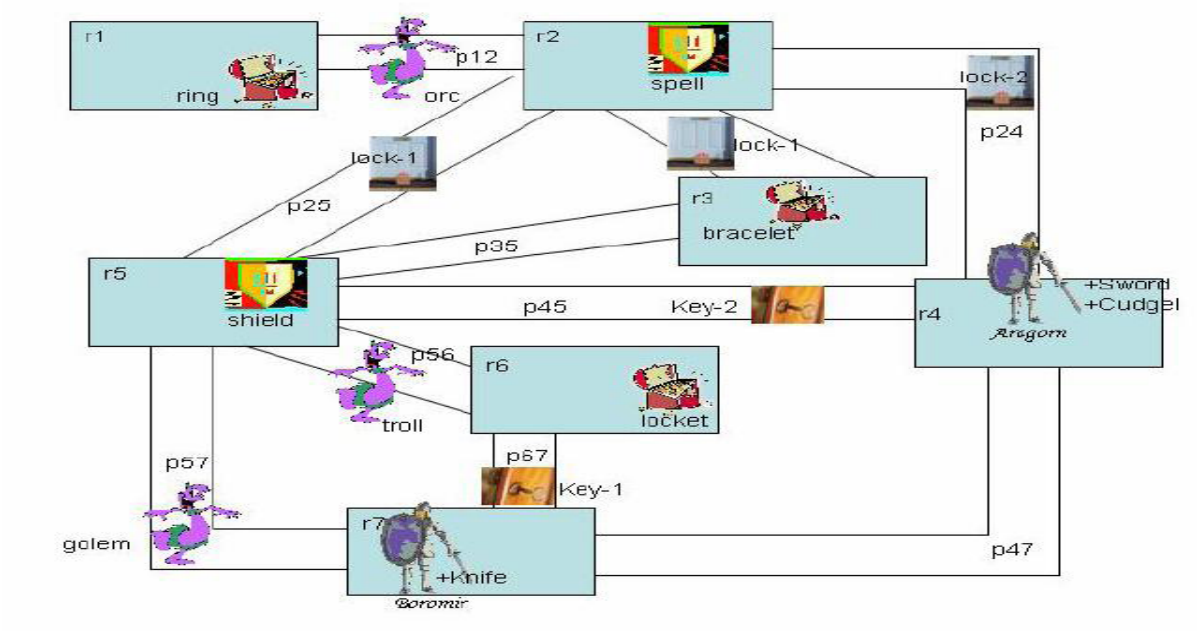


Figure 1: A state of the Ring World

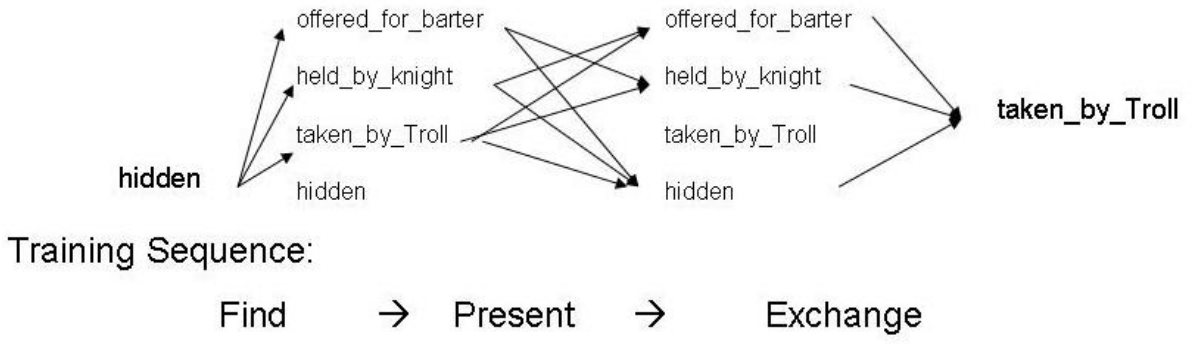
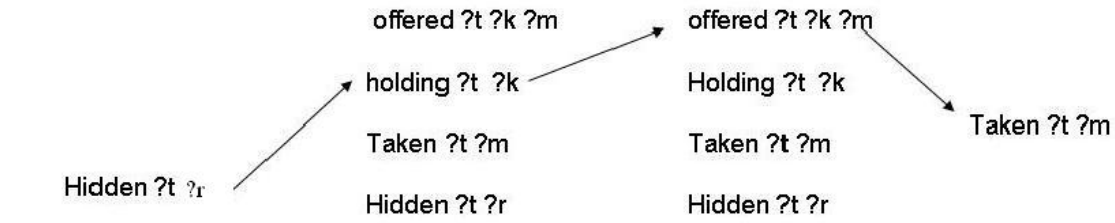


Figure 2: Seven paths through object states



Training Sequence:

Find_treasure ?t ?r ?k → Present ?t ?m ?k → Exchange ?t ?k ?key ?m

Figure 3: A unique path through object state