On Comparing Planning Domain Models

S. Shoeb and T.L.McCluskey,
Department of Informatics,
School of Computing and Engineering,
University of Huddersfield

Abstract

Whereas research into the characteristics and properties of AI planning algorithms is over-whelming, a similar science of domain model comparison and analysis is underdeveloped. It has long been acknowledged that there can be a range of different encodings for the same domain, whatever coding language used, but the question of which is the "best" encoding is an open one, and partly dependent on the requirements of the planning application itself. There is a growing need to measure and compare domain models, however, in particular to evaluate learning methods. In this paper we motivate the research by considering the challenges in evaluating domain model learning algorithms. We describe an ongoing doctoral research project which is exploring model classifications for comparing domain models.

Introduction

The importance of the knowledge formulation process to the success of planning applications, and to make planning engines more accessible and open to community use, is now well established. The task of creating the domain models is accepted as difficult and time consuming, requiring an engineering process. This has increased the need to investigate the properties of the end product of this process - the planning domain model. It has long been acknowledged that there can be a range of different encodings for the same domain, whatever coding language used, but the question of which is the "best" encoding is an open one, and partly dependent on the requirements of the planning application itself.

The advances made in automated domain model creation, in particular, have focused effort on such questions as what makes an accurate or efficient domain model. In this area, domain models may be learned from examples (Cresswell, McCluskey, and West 2011; Wu, Yang, and Jiang 2005), learned from one example plus a partial domain model (McCluskey, Richardson, and Simpson 2002; McCluskey et al. 2010) or generated from a formal model using a translation process (Ferrer 2011; Bartak, Fratini, and McCluskey 2010).

Domain analysis techniques can of course be used to inform on the quality of a domain model, and in particular whether the model is self-consistent. They are well developed for AI planning, and can be used to check certain desirable static features such as operator reversibility: that is whether at any state, an operator's application can be reversed by another operator to result in the original state (Wickler 2011). Another use is to analyse the likely efficiency of a model when paired with a state of the art state progression planner. We may be interested in knowing the "landscape" of the key heuristic used in planning, such as investigated in Hoffman's recent research (Hoffmann 2011).

The steady increase in the development of domain model learning tools and the need to extract a healthy domain model from such tools requires the development of measurement techniques to evaluate a domain model. One way to do this is to quantify certain features that can easily be calculated and compared: the number of objects, the number of action instances, the degree of arity of operators names, the degree of arity of predicates and the average number of preconditions and postconditions in each operator.

In our research, we consider that the quality of automatically produced domain models needs to be evaluated in order to provide validation and comparison between competing automated approaches. This is of key significance for researchers attempting to evaluate their research in a rigorous fashion. Of course the quality of a model can be measured in a dynamic way, that is by running the models with a planner, and checking performance and output plans. This is a reasonable approach, but has a major flaw: performance is of course dependent on the planner as well as the model, and the results may be due to a planner rather than a model. Ideally, we would also like to measure and compare domain model in a planner-independent way. In this paper we motivate the reader as to why comparison techniques are important, and make some initial definitions of equivalence between domain models.

Motivating Examples

LOCM is an operator schema learning system (Cresswell, McCluskey, and West 2011) which automatically induces a set of operator schema from sets of example plan scripts. LOCM assumes that each plan is a sequence of ordered actions, and each action is composed of a name and a list of objects. The system differs from related systems in that it requires no other knowledge to be provided (i.e no predicate structures, or initial and intermediate states, are required for LOCM to function). LOCM works well under the assumption that objects in a domain can be modelled as going through transitions in a object-centered state machine, and from inducing these state machines LOCM produces PDDL domain models.

One method of evaluation of LOCM is as follows: using existing hand crafted models, generate sets of plan traces. Feed these plan traces into LOCM and use it to output a domain model, then compare this output model with the original hand crafted.

The first challenge in this form of evaluation is to create a mapping between parts of the two models (the original and the learned). LOCM learns predicates, so that the 'anonymous' predicates in the learned model have to be matched up with those in the original model. A consistent mapping has to be made from the names in the learned model to the (meaningful) names in the original. In some domains, LOCM may generate domain models that, under such a name mapping, behaviourally seem equivalent to the hand-crafted model.

For example, consider the case when LOCM attempts to learn the "tyre domain" model using the original hand crafted model from which to generate plan script examples. Under the correct name mapping, a task (goal and initial state) in the notation of the original domain model can be mapped to the learned model notation; given to a planner, the output produced will be the same as if the original model was used. But consider the predicate definitions in the fragment below, which shows part of the hand-crafted model concerning "jack" related predicates, and the corresponding predicates in the induced model:

Hand Crafted:

```
(jack_in_use ?jack1 - jack ?hub1 - hub)
(have_jack ?jack1 - jack)
(jack_in ?jack1 - jack ?boot1 - boot)
```

Induced:

```
(jack_state0 ?v1 - jack ?v2 - hub)
(jack_state1 ?v1 - jack ?v2 - hub)
(jack_state2 ?v1 - jack ?v2 - hub)
(jack_state3 ?v1 - jack)
(jack_state4 ?v1 - jack ?v2 - boot)
```

The induced model contains 3 variations of "jack X in use on hub Y" predicate. Each variation records a different state of the hub assembly: when it is jacked up with the hub free, or with a loose wheel attached, or with a wheel fastened on. In this case, the learned model is certainly "close" to the original — behaviourally the two models appear the same, though the induced model is more fine grained; it may even be argued the induced one is better. The problem is, how can we compare and measure systematically such differences, and hence evaluate the learning mechanism?

LAMP (Zhuo et al. 2010) is a successor of the ARMS algorithm (Wu, Yang, and Jiang 2005), and designed to learn complex structured domain models containing quantifiers and logical implications. LAMP requires as input a set of observed plan traces, a list of actions composed of names and parameters, and a list of predicates with their corresponding parameters. It is aimed at learning in a mixed initiative fashion: the authors acknowledge that a model learned by LAMP needs to be refined by experts in order to produce a final domain model.

In the evaluation of LAMP, the authors use a very simple metric: they define the error rates of the their algorithm as the number of different predicates in either the preconditions or the effects of an operator schema, as a proportion of the total number of predicates. In the LOCM tyre example, this would result in a non-zero error rate, since there are 2 extra predicates different in the induced model (under some reasonable name mapping), and these appear in several of the operator schema.

If for example, it was required to show that with more examples, LAMP would be more accurate, then it would be necessary to show that one model was "nearer" the hand crafted version than another. Even with predefined predicates, it could be argued that the evaluation of LAMP is problematic in that is it not obvious how "near" the models it produces are using such syntactic distinctions. As the example in LOCM shows, this measure does not seem appropriate in the situation where the system learns predicates. We need measures that obey certain rules — for example a measure that is monotonic in the sense that as the error rate was lower, the domain created was "nearer" the original domain.

As well as comparing domain models for evaluation, we may want to compare or evaluate domain models as an aid to learning a new model. In the LAWS system (H.H.Zhuo, Q.Yang, R.Pan and L.Li 2011), the idea is to use a pre defined domain model from which to learn a model of a similar domain, in much the same way as a human might re-use an old model and adapt it to a new domain. Here again the concept of comparing models for their similarity is an important one, as the source domain has to be close to the target domain in some sense.

Model Comparison Definitions

The research we are carrying out is exploring properties and classifications to use in comparisons of domain models. We are utilising the wealth of past research on domain (model) analysis, and basing our ideas on our earlier investigative work (McCluskey 2003; McCluskey, Richardson, and Simpson 2002).

Strong Equivalence

We define **strong equivalence** where two domain models are strongly equivalent iff they are logically identical up to naming. To establish such an equivalence, a 1-1 mapping must exist which maps all the names in one domain to another (names of predicates, variables, operator schema, types). After a mapping has been performed, it is assumed that ordering of declarations of types, predicates and actions does not matter. “Logically Identical” here means that for each action A1 in one domain that maps to an action A2 in the other domain, A1.pre is logically equivalent to A2.pre, and A1.effects is logically equivalent to A2.effects. A good example is a translation of names in a domain model from English to French. The resulting model with names in the French language is strongly equivalent to the one in English, and vice-versa. We could perform further changes to the domain model by changing the order of declaration of types and predicate, or changing the logical conditions in the domain model using properties of logical connectives etc. In these cases the two domains would remain strongly equivalent.

Weak Equivalence

We can define **weak equivalence** for models analogous to “weak equivalence” in grammars — meaning two grammars generate the same language. The grammars may not be structurally the same, but they share the same behaviour. Thus, we say two domain models A and B are weakly equivalent iff there is a domain model B’ which is strongly equivalent to B, such that:

- any task (I,G) that can be posed in A can be posed in B’, and vice-versa
- for any task (I,G), any complete and corrent plan that can solve it using model A is a correct and correct plan according to B’, and vice-versa.

Weak equivalence in domain models therefore means that the functional behaviour of the domain is the same for both models – the same tasks can be formulated, and the same solutions can be generated. In the LOCM example, the two domains are not weakly equivalent as the mapping between predicates in not 1-1. In this case, any tasks posed in the original can be solved in the induced model, but not the other way round.

Summary

In this paper we have highlighted the need to research into sound methods for comparing and measuring domain models. We have used two compelling examples of systems that learn domain models to illustrate the problem, and have started to develop a framework for model comparison.

The research questions we are investigating include (a) what are useful classifications of similarity of domain models? (b) can we produce support tools for checking similarity? (c) what are good metrics for domain models, and are they useful for comparing domain models?

References

Bartak, R.; Fratini, S.; and McCluskey, L. 2010. The third competition on knowledge engineering for planning and scheduling. AI Magazine, Spring 2010.

Cresswell, S.; McCluskey, T.; and West, M. M. 2011. Acquiring planning domain models using LOCM. *Knowledge Engineering Review (To Appear).*

Ferrer, A. G. 2011. *Knowledge Engineering Techniques for the Translation of Process Models into Temporal Hierarchical Planning and Scheduling Domains.* Ph.D. Dissertation, Universidad de Granada.

H.H.Zhuo, Q.Yang, R.Pan and L.Li. 2011. Cross-Domain Action-Model Acquisition for Planning Via Web Search. In *Proceedings of ICAPS.*

Hoffmann, J. 2011. Analyzing search topology without running any search: on the connection between causal graphs and h+. *J. Artif. Int. Res.* 41:155–229.

McCluskey, T. L.; Cresswell, S. N.; Richardson, N. E.; and West, M. M. 2010. Action Knowledge Acquisition with Opmaker2. In *Agents and Artificial Intelligence*, volume 67 of *Communications in Computer and Information Science.* Springer Berlin Heidelberg. 137–150.

McCluskey, T. L.; Richardson, N. E.; and Simpson, R. M. 2002. An Interactive Method for Inducing Operator Descriptions. In *The Sixth International Conference on Artificial Intelligence Planning Systems.*

McCluskey, T. L. 2003. PDDL: A Language with a Purpose? In *Proc. PDDL Workshop, ICAPS, Trento, Italy.*

Wickler, G. 2011. Using planning domain features to facilitate knowledge engineering. In *Proc. KEPS Workshop, ICAPS.*

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.*

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artificial Intelligence* 174(18):1540–1569.