Learning Action Models for Planning: An Overview of the Hedlamp Project

Thomas Leo McCluskey and **Lukáš Chrpa**
PARK Research Group
School of Computing and Engineering
University of Huddersfield
{l.chrpa, t.l.mccluskey}@hud.ac.uk

Austin Tate and **Gerhard Wickler**
Artificial Intelligence Applications Institute
School of Informatics
University of Edinburgh
{a.tate, g.wickler}@ed.ac.uk

Abstract

Hedlamp is a UK EPSRC grant funded research project in which we aim to tackle challenges with knowledge engineering of automated planning techniques when applied to real applications. Normally, successful deployment of planning technology relies on groups of planning experts encoding detailed domain models and investing large amounts of time maintaining them.

We are developing a high level, application-oriented knowledge engineering framework usable by application developers who want to experiment with the potential of AI Planning, while encoding a precise domain model of some valuable application area. We are developing tools and theory for the framework which support knowledge acquisition, validation and operationality of the domain model. In particular, this project aims to explore the opportunities of applying model translation, adaptation and reformulation techniques to improve the model's quality, and that of the planning function of which it is a part.

In this paper we outline the main areas that Hedlamp has addressed, and overview an automated knowledge acquisition technique that has been tested with real industrial process data.

Introduction: Project Context

Hedlamp (Huddersfield and Edinburgh: Learning Action Models for Planning) is a UK EPSRC joint grant funded research project, 2012-16, in which we in which we aim to tackle challenges with knowledge engineering of automated planning techniques. Our work is carries on from previous knowledge engineering research platforms such as GIPO (Simpson, Kitchin, and McCluskey 2007) and ItSimple (Vaquero et al. 2012), but is also heavily influenced by the EPSRC AIS Programme's industrial setting[1]. Utilizing planning machinery relies on the expertise of groups of planning experts having to learn application expertise, encoding detailed domain models of the relevant part fo the application, and investing large amounts of time maintaining the model. We are developing theory and techniques to meet the challenge through involving the expert in the encoding of the knowledge, developing V & V techniques which identify and remove bugs at the domain modelling stage, developing techniques to acquire the knowledge automatically, and optimising the translation of the domain model into a form acceptable by a planning engine. The tools are built around the key concepts of expert involvement and inspection, translation, reformulation, and machine learning, This has led to the development of a high level, application oriented knowledge representation language (AIS-DDL), with translators to the family of PDDL planner input languages, as well as the development and use of an hierarchical planner which inputs AIS-DDL. Given planning knowledge is notoriously hard to encode, we need effective ways of removing bugs and certifying its quality. The interface needs multiple ways to perform V & V to enable this. Following this rationale, our progress has largely covered the following areas:

- manual coding of an AIS programme application, expert involvement and validation: the knowledge representation lanaguage AIS-DDL has been developed such that the expert user can both add to it and inspect it. This involves it being accessible on a networked platform called "KEWI", which includes documentation and (links to) source material for traceability (G.Wickler, L.Chrpa, T.L.McCluskey 2014a).
- parsing and translation: components of AIS-DDL are checked, parsed and translated down to an appropriate PDDL level for input to a planning engine. The translator embodies consistency checking, and the operational form is run to help in validation, hence V & V processes are being addressed (G.Wickler, L.Chrpa, T.L.McCluskey 2014b).
- reformulation and heuristic learning: given the translation is automated, the resulting PDDL is not optimised and could include inefficient components. In this case, problem transformation components are useful to input the translated PDDL and transform it into a more efficient representation, as well as deriving heuristics in the form of macros. The transformations preserve the initial model's semantics and keep it within the same PDDL language. We have shown that these techniques are both domain and planner independent (L.Chrpa, M. Vallati, T.L.McCluskey 2014).
- internal planner: For validation by operation, we have developed a hierarchical planner within KEWI. This has the advantage of operational heuristics embodied within AIS-

[1] https://www.epsrc.ac.uk/files/funding/calls/2011/autonomous-and-intelligent-systems/

DDL which correspond to known procedures or activities involving a collection of primitive actions.

- machine learning: where other sources of data can be identified (apart from expert involvement and the use of expert documents), then it is possible to consider the use of machine learning techniques either in real time or batch mode. For example, work in the area of ATC used tracking records to perform machine learning in the form of theory revision on an existing domain model (McCluskey and West 2001). From initial acquisition and domain maintenance viewpoints, the use of training data can lead to the automated reconstruction or evolution of the domain model.

A central element of the HedLamp research project is in its aim to develop procedures to *automatically learn domain models* for automated planning. In this report we will give an overview of the recent developments in adding acton model learning to the KEWI environment, in particular to benefit from data output from inductrial processes. First, we will survey related work along these lines.

Learning Domain Models for Planning

Overview

Machine Learning applied to *APS* has attracted a long history of research, and can be applied to various areas of automated planning such as learning heuristics. Here we concentrate on learning applied to knowledge acquisition of action schema, and point the reader to a recent survey for a full account (Jiménez et al. 2011). Using techniques from the field of Machine Learning, researchers have experimented with processes that input training or observation inputs, and output solver-ready models in languages such as PDDL. To be able to do this, the learning processes have to embody various types of knowledge such as general properties and constraints about actions and objects, as well as knowledge about the kind of domain in which they are learning. Motivations for this type of learning include the following interconnected challenges: to overcome the knowledge acquisition bottleneck; to alleviate "coding bias" of knowledge engineers; to debug existing doman models; and to be able to apply planning to adapted or new domains without re-coding of models.

As an example, the knowledge engineering environment GIPO III (Simpson, Kitchin, and McCluskey 2007) embodied an induction technique to aid the acquisition of operator descriptions, called *OpMaker*. The tool requires as input a structural description of the static parts of the domain comprising knowledge about states of objects and their relations. Given a training problem instance and a valid solution plan for that instance, *OpMaker* outputs a full PDDL model (McCluskey et al. 2010). This kind of domain model learning can be separated into three concerns:

1. What inputs (training plans, observations, constraints, partial models, etc.) are there to the learning process?
2. What stage in the development process is the learning taking place (initial acquisition, or incremental, online adaptation)?
3. What language is the learned domain model going to be expressed in?

In the case of *OpMaker*, (i) is a partial model and one example, (ii) is initial acquisition, and (iii) is PDDL version 2.1. Several recent notable learning systems (OpMaker, LAMP, LOCM) aim to output some variant of PDDL within an initial acquisition phase. While sytems tend to concentrate on initial acquisition, adaptation can be viewed itself as a non-monotonic special case of initial acquisition, where input to the learning process includes the current domain model as well as training examples and the output is the updated model. Wang's OBSERVER system was a seminal example of this, as it learns an initial model and continues to perform repairs to the model during operation (Wang 1995).

Regarding (i), systems that learn very expressive domain models tend to demand the most detailed input, often requiring detailed state information before and after action execution within each training plan. Past work on learning domain models for robotic agents in uncertain environments assumes such detailed input and substantial *a priori* knowledge (Amir 2005; Benson 1996). With such rich inputs, systems such as Amir's SLAF (Amir 2005) can learn actions within an expressive action schema language.

In comparison, significant recent work on learning domain models within a deterministic and totally observable framework has concentrated on learning from example plans but with little or no pre-engineered domain knowledge. The LAMP system (Zhuo et al. 2010) can form PDDL domain models from example plan scripts and associated initial and goal states only. It inputs object types, predicate specifications, and action headings, and from plan scripts taken from planning solutions, it learns a domain model. The domain model is synthesised using a constraint solver, inputting two sets of constraints. One set is based on assumed physical, consistency and teleological constraints – for example, every action in the example plan script adds at least one precondition for a future action, actions must have non-empty effects, and so on. The other set of constraints is generated using a type of associative classification algorithm which uses each plan script as an itemset, and extracts frequent itemsets to make up constraints. LAMP is aimed at helping knowledge engineers create a new domain model, as the authors maintain that, after learning, the model needs to be handcrafted to remove bugs. Another family of systems rooted in LOCM, exploit the assumption of an object-centred domain to enable learning from plan scripts only (Cresswell, McCluskey, and West 2011). As with LAMP, LOCM outputs a model in a PDDL format but it inputs *only* training plan scripts: it does not require representations of initial and goal states, or any descriptions of predicates, object classes, states etc. LOCM assumes that the objects referred to in the training plans can be clustered into classes, and each class has a behaviour defined by a parameterised state machine, which it constructs using implicit physical constraints on the state change of objects. While being a useful induction engine, LOCM also, to some degree, avoids the encoding bias of knowledge engineers mentioned above.

To be successful, automated tools that learn domain mod-

els have to embody general properties and constraints about actions and objects, and in most cases the kind of domain in which they are learning. The key idea within these approaches is that of *inductive generalization* – using examples of behaviours of a class of objects and generalising these examples to a theory about the whole class of objects. In the case of planning, a set of plans that are observed from the domain itself is a natural training input. Potential training plans could be harvested from a wide range of applications, and examples include logs of commands such as operating system instructions, moves made in a game or traces of work-flow or business process execution - any output of some operation in the application domain from which we can extract information relating to the domain model.

An Overview of Action Model Learning in KEWI

The HedLamp project was aimed at supporting autonomous intelligent systems in the industrial sectors supporting the EPSRC AIS Programme. Hence, we worked on knowledge acquisition and engineering of knowledge of industrial processes, supplied by the industrial backers of the Programme.

The target of learning was action schema, both to help the system adapt over time, and to provide another form of validation of previously encoded schema. Referring to the three concerns above, the output (iii) is the KEWI source lanuage (AIS-DDL), and the stage of learning (ii) is post initial acquisition. Inputs (i) to the learning process are detailed: applications where knowledge engineering is required to capture the objects and their dynamics in an area such as industrial plant operation, there is a great deal of persistent structural and constraint knowledge which is naturally captured by encoding with the help of domain experts and domain manuals. Additionally, some of the knowledge may already exist in ontological form, and may be translatable to AIS-DDL. Hence, the kind of learning method we will need would take account and indeed benefit from existing knowledge. We are interested in those techniques, therefore, which assume that part of the input to learning process is a partial domain theory, which includes some of the previously encoded model. The other question is: what data is available to drive the learning taking place? In scenarios describing some external controllable process such as industrial plant operation, the inputs to a learning component are data traces of processes which contain objects of classes already encoded in KEWI. These traces are typically observations of the state of features and properties of the equipment - in other words, state data in a raw form.

The nature of the problem area therefore determines the kind of learning technology we need to use: we have a partial domain model, we have a stream of raw data, and we would like to learn action models in AIS-DDL. In this case, our solution would be inspired by systems such as Opmaker (McCluskey et al. 2010) or LAMP (Zhuo et al. 2010), where we are learning action models from state information, and previously encoded "static" knowledge. Before we describe the learning process, we will give an overview of the language in which the partial domain model is encoded, and in which the output will be formed.

The KEWI Source Language

KEWI (G.Wickler, L.Chrpa, T.L.McCluskey 2014b) was designed to help the formalisation of procedural knowledge both for use in planning, as other knowledge based applications such as plan explanations. The idea is to enable domain experts to encode knowledge themselves, rather than using knowledge engineers. KEWI is object-centred and allows for a richer representation of knowledge than PDDL. It is more compact and more expressive which means models are easier to maintain, especially for a user who is not an expert in AI planning. KEWI's internal representation can be exported to PDDL (though information will be lost in the process) and hence standard planning engines can be applied to solve planning problems modelled in KEWI. Domain knowledge encoded in KEWI is at three levels: level 1: a rich domain ontology is defined. The domain ontology consists of definitions of classes of objects, hierarchies of classes and relations between objects. Level 2: action types are defined in terms of their action name, logical preconditions and effects. Level 3: methods define ways in which high level task can be broken down into lower-level activities, a so-called task network which includes explicit ordering constraints. Ontological elements are usually divided into concepts and instances. Typically, the concepts are defined in a planning domain whereas the instances are defined in a planning problem.

For example, part of "level 1" AIS-DDL for the "dock workers" domain, used in the Planning textbook (Ghallab, Nau, and Traverso 2004), is as follows:

```
(:domain dwr
  (:class agent)
  (:class crane
    (:super-class agent)
    (:role at (:min 1)(:max 1)(:class location))
    (:role holds (:max 1)(:class container)))
  (:class robot
    (:super-class agent)
    (:role loaded-with (:max 1)(:class container))
    (:property has-colour (:min 1)(:max 1)
       (:type colour)))
  (:class location
    (:role occupied-by (:max 1)(:class robot)))
  (:class stackable)
  (:class container
    (:super-class stackable)
    (:role on (:max 1)(:class stackable))
    (:role piled-on (:max 1)(:class pallet))
    (:property paint (:min 1)(:max 1)
       (:type colour)))
  (:class pallet
    (:super-class stackable)
    (:role at (:min 1)(:max 1)(:class location))
    (:role top (:min 1)(:max 1)(:class stackable)))
  (:property colour
    (:values ( red green blue )))
  (:relation adjacent
    (:arguments ((?loc1 location)(?loc2 location)
    )))
```

Learning Action Models from Raw Data

We detail the model learning method, and its application to a real process plant assembly. The industrial process delivers to us raw values / measurements of objects *which are already encoded within a KEWI class*. This means that the definition of the features and properties of objects that the data is describing (e.g. status of a pump) are already declared within KEWI. We therefore have taken a staged approach:

1. collection of raw data from the operation of the mechanics of the system that is already (partially) encoded within KEWI;
2. translation of the trace of the raw data into symbolic data sequences, where the symbols are defined within KEWI, and each element of the sequence describes a partial state;
3. inducing/deducing action schema/models from the symbolic data sequences

The first two stages are application dependent, though tied, of course to the ontology supplied by the knowledge in KEWI. This kind of translation has been carried out in much previous work, for example, mapping from geometric states to symbolic states in robot maniputation applications (Dearden and Burbridge 2014), or low level sensor data to predicates in action learning (Laguna 2014). In our work, we place the previously engineered knowledge as central to the type of learning performed: the blend of induction with deduction in stage 3 is key to the success of the method.

We created a domain-dependent procedure to discretize the continuous streams of sensory inputs, dealing with the noise in the data, and building on known dependency between values. In particular, noise can be in form of missing or incorrect values. Whereas it is easy to deal with missing values, (for instance, by ignoring the data item), incorrect values might not be straightforward to identify. This is because a value that differs from the previous ones might be either inaccurate or indicate a change of a state. For example, a stream of values representing the number of rotations per minute of some movable part of a plant assembly determines whether the part is rotating or not (i.e., the values are discretized into a binary state). A sequence of zeroes in that stream thus indicates that the part is not rotating. However, if a non-zero value occurs we have to decide whether the state of the part has changed to "rotating", or the value is incorrect. We cannot draw conclusions from a single value, however, we can observe n values (we set $n = 10$) and majority of the values indicates a change of the state (e.g. if 8 out of 10 values describing the part's rotation are non-zero, then the state is changed to "rotating"). A similar approach is used to determine state changes for states that are determined from multiple different sensory inputs (e.g. such as an exogenuos event that causes direct or indirect changes in the plant). In such cases, the noise is considerably larger. To overcome this issue we set up specific rules for each state value change (e.g. changing between the mode of operation of a moveable part of an assembly).

For the third stage, we assume that the sets of values making up each state in the trace of the system are known to describe observations (here values of properties) of parts of a known object in KEWI (here parts of the plant assembly). Hence a sequence of object feature variable - values is input to the third stage. We base the method of the third state on two ideas:

- Deduction - we use a priori knowledge to enrich learned patterns
- Inductive generalisation - we induce patterns from many examples

In the deduction step, we use the fact that object knowledge within the domain model contains the specification of the set of states of a class of objects. For example, recalling our example in the Dockworkers domain, for a *Container*, it specifies that there are 3 legal states that an object of class container may occupy. Each state is represented by one of the three predicate sets below:
–paint(Container,Colour),on(Contained,Stackable),
piled_on(Container,Pallet)
–paint(Container,Colour),on(Contained,Stackable)
–paint(Container,Colour), piled_on(Container,Pallet)
Hence, as objects are affected by sequences of actions, they potentially travel through the state space as specified in the static model, and objects which are Containers are therefore in one of the three states as specified above, in any state as the plan progresses. The inductive generalisation part incorprates these constraints from KEWI's 'level 1' knowledge base. It uses the many examples of state changes, analyses the context of each change of state property, and induces action schema, as outlined below:

1. Iterate through the whole batch of input and record each object feature change
2. For each feature variable-value change:
   (a) collect the pre- and post values of other variables in the before/after states;
   (b) induce event pre-conditions : gather variables with a fixed value for each instance of change
   (c) induce post-condition: record cases where other values change at the same time
   (d) create parameter list from objects involved in the transition

A change is where the variable's value changes from one state in the sequence to the next (there are sometimes more than one change). A list of KEWI 'level 2' operators are output from this process.

Results

We have implemented and tested the model learning method on a data sequence of length c.60,000 records over c.2 days of operaton of a real industrial process, with 23 value readings per record. The learning system generates intermediate states, then learns action and event schema in KEWI syntax. Of the 16 schema learned, 14 represented new actions or events, in the sense that they were not already encoded into KEWI. The 2 hand crafted actions already present in KEWI were not inconsistent from the 2 counterpart learned versions, in that the latter were more specific and formed

wth extra preconditions. For example the following action schema was induced by the system:

```
(:action-class off-to-on-pump
    (:arguments
        ( (?x pump)(?x1 plant-assembly) ))
    (:precondition (:and
     (:constraint
        pump.state ( ?x off))
     (:constraint
        plant-asembly.main-activity
           ( ?x1 inactive))))
    (:effect (:and
     (:constraint
       pump.state ( ?x on )))))
```

This introduces a new precondition that was not recognised - that the plant assembly's main activity must be inactive before the pump is activated. This raises validation issues with the original encoding. An interesting side effect of the method is that the original raw data can be replaced by events and actions throughout the time period, replacing raw traces with sequences of events and action applications, giving a high level description of the happenings over the 2 days. The induction engine creates several hundred of these events and actions that occur over the period - a small snapshot of the sequence is shown below, where at two stages two actions/events occur simultaneously [2]:

```
disengage-assembly
stop-rotating   stop-pump
start-pump   suspect-overload
start-rotating
no-overload-found
start-engaging-assembly
```

The system is integrated with KEWI in the sense that it uses translated KEWI knowledge structures to inform it, and produces KEWI action descriptions. The results have been presented to domain experts, but we are yet to perform a full analysis, or (automatically or otherwise) integrate the new knowledge into the KEWI knowledge base, and deal with induced schema that are different from previously acquired actions.

In comparison to existing systems, KEWI's learning component can be seen as a generalisation of Opmaker, as it utilises a priori domain knowledge in the form of the Level 1 partial domain model to direct learning. Additionally it incorporates into it an inductive part, so that the process benefits from many examples; and its inputs are derived from real data, rather than simulated.

Conclusions

In this paper we have given an overview of an industrial-oriented EPSRC project which is aimed towards tackling challenges in knowledge engineering for automated planning applications. We outline the wide range of knowledge engineering techniques that the research has been aimed at. The paper looked in more detail at the area of learning action models, and described such an implemented and tested system. Our current work is engaged on the integration of the learned knowledge, with the existing knowledge, in the KEWI knowledge engineering platform.

[2] names for actions, events, processes etc have been made generic in this paper so as to protect the privacy of the specific industrial sector involved

References

Amir, E. 2005. Learning partially observable deterministic action models. In *Proc. IJCAI 05*, 1433–1439.

Benson, S. 1996. *Learning Action Models for Reactive Autonomous Agents*. Ph.D. Dissertation, Stanford University.

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

Dearden, R., and Burbridge, C. 2014. Manipulation planning using learned symbolic state abstractions. *Robotics and Autonomous Systems* 62(3):355 – 365.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kaufmann ISBN 1-55860-856-7.

G.Wickler, L.Chrpa, T.L.McCluskey. 2014a. Creating Planning Domain Models in KEWI. In *Proceedings of KEPS, 24th International Conference of Automated Planning and Scheduling, Portsmouth, New Hampshire*.

G.Wickler, L.Chrpa, T.L.McCluskey. 2014b. KEWI: A Knowledge Engineering Tool for Modelling AI Planning Tasks. In *Proc. 6th International Conference on Knowledge Engineering and Ontology Development, Rome, Italy*.

Jiménez, S.; De la Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2011. A review of machine learning for automated planning. *The Knowledge Engineering Review*.

Laguna, J. O. 2014. *Building Planning Action Models Using Activity Recognition*. Ph.D. Dissertation, Universidad Carlos III de Madrid, Spain.

L.Chrpa, M. Vallati, T.L.McCluskey. 2014. MUM: A Technique for Maximising the Utility of Macro-operators by Constrained Generation and Use. In *Proceedings of ICAPS 2014, AAAI Press*.

McCluskey, T. L., and West, M. M. 2001. The automated refinement of a requirements domain theory. *Journal of Automated Software Enginnering, Special Issue on Inductive Programming* 8(2):195 – 218.

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.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007. Planning Domain Definition Using GIPO. *The Knowledge Engineering Review* 22(1).

Vaquero, T. S.; Tonaco, R.; Costa, G.; Tonidandel, F.; Silva, J. R.; and Beck, J. C. 2012. itSIMPLE4.0: Enhancing the Modeling Experience of Planning Problems. In *Proceedings of ICAPS 2012 System Demonstration. Atibaia, Sao Pualo, Brazil*, 11–14.

Wang, X. 1995. Learning by Observation and Practice: An Incremental Approach for Planning Operator Acquisition. In *Proceedings of the 12th International Conference on Machine Learning*.

Zhuo, 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.