A Brief Review of Automated Knowledge Engineering Tools for Artificial Intelligence Planning & Scheduling

R. Jilani and D. Kitchin
University of Huddersfield, Queensgate, Huddersfield HD1 3DH, UK

ABSTRACT

It needs much effort to encode a planning domain model for complex applications. It includes getting the domain knowledge, creating and validating the correctness of knowledge and then maintaining it. The major product is the domain model which consists of a set of operators [1] (also called action schema) written in certain modelling languages. In this paper we briefly review a variety of automated knowledge engineering tools that automatically produce domain models, and discuss the motivation behind the development of these tools with three major concerns that differentiate these tools from one another. The results of the review give insights into the strengths and weaknesses of the considered systems, and point to needs in future work to enhance the capabilities and overcome their short comings.

Keywords Domain model, knowledge engineering, artificial intelligence (AI) Planning

INTRODUCTION

Knowledge is a theoretical or practical understanding of a subject or a domain. Knowledge is also the sum of what is currently known, and apparently knowledge is power [2]. Those who possess knowledge are called experts. And the process by which we gather, formulate, validate and input information, in the form of knowledge into computer systems, is called Knowledge Engineering (KE). Knowledge engineering provides a huge database of information to computers to draw knowledge from to answer complex problems and automatically design new plans for proposed problems without any intervention by humans. This includes the study of application requirements and creating a model that explains the domain and then testing its correctness. Knowledge engineering has become a vital research area in many walks of life e.g. robotics, transport logistics, web-based information gathering and game software. Beside a problem description, Automated Planning (AP) engines need a huge knowledge reservoir containing the world knowledge about the domain (domain knowledge). This is where knowledge engineering and automated planning shakes hand. Here huge reservoir of knowledge about the required domain is provided by knowledge engineering tools, as domain knowledge.

Encoding a planning domain model is a difficult task in complex applications. It includes the study of planning application requirements, creating a model that explains the domain, and testing it with suitable planning engines [3]. Knowledge engineering tools for automated planning have been used in designing new manual and automated systems and interfaces that generate knowledge about actions and how objects are used by actions. This knowledge has to be satisfactory to allow efficient automated reasoning and construction of a plan as output.

These tools include some manual as well as automated tools. Manual tools include itSIMPLE [3], GIPO [4], JABBAH [5], EUROPA [6], MARIO [7], VIZ [8] and PDDL Studio [9]. Automated tools

includes ARMS [10], Opmaker [11], Opmaker2 [12], RIMS [13], the system of Shahaf and Amir [14] and LOCM (Learning Object Centred Models) [15, 16]. There have been reviews of manual knowledge engineering tools and techniques for AI Planning in [17] and [1].

One approach to the problem of automatically formulating domain models for planning is to learn the models from example action sequences. Keeping this approach in mind we aim to present a brief overview of the automated Knowledge Engineering tools that can learn from provided plan traces and improve system performance with experience [7] for automated planning application models. The plan traces or training data consist of a set of training examples. It “learns” or “observes” training plans containing action instances and “exploits assumptions about the kinds of domain model it has to generate, rather than using handcrafted clues or planner-oriented knowledge” [18]. Based on good development in research on automated tools for knowledge engineering in Artificial Intelligence (AI) planning, Tiago Stegun Vaquero et. al. in [17] conclude that knowledge engineering is better introduced in the planning world through the domain design process, more than through the planning techniques. Thus, we examine the fundamental methodologies and techniques in the development of each of different knowledge engineering tools.

General Motivation behind Automated KE Tools

There are several motivations behind the need for automated knowledge engineering tools:

1. To help planning agents become more autonomous and make them capable to adapt and plan in unseen domains they must be able to learn a new domain model.
2. To make intelligent agents able to correct/refine already generated domain models and teach themselves to grow and change. It is difficult to generate correct a domain model at the first attempt so agents must be able to incrementally adjust their existing model to deal with unexpected changes.
3. Last but not least, Knowledge engineering of complex domains manually by hand into planning languages such as PDDL is an inefficient, tedious and error-prone task, because it needs a human, expert in domain knowledge, to type in action schema, which can take hours and could also cause unintentional errors. So human-driven knowledge acquisition and maintenance tasks requires automated support if not fully automated.

DISCUSSION

All KE automated tools that learn models from example plans differ in the amount of the additional input information required to generate a domain model. Additional information could be full or partial knowledge about predicates, initial, intermediate and goal states, and some of these may need a partial domain model as input to generate a full domain model as output. These systems also differ in the representation of the formalism in which action schema are formed in output.

There is already a lot of research aimed at automatically learning and maintaining a useable domain model for the purpose of high level reasoning, specifically to support AI planning [19].

Generally automated domain model learning can be separated into the following three concerns [19]:

i) In what language will the domain model be generated?

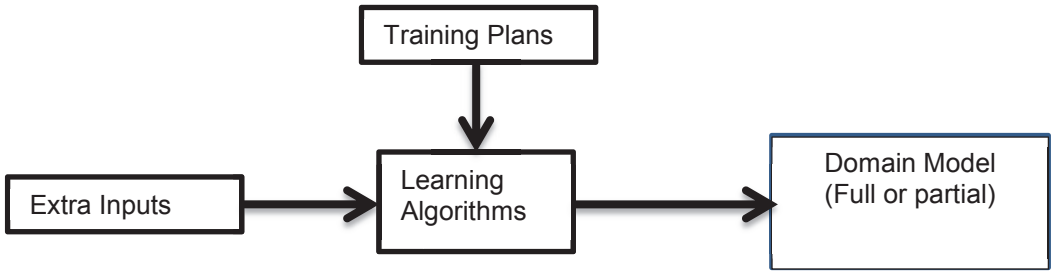
Based on most of the research done on domain model generation, PDDL (Planning Domain Description Language) and its variants are commonly being used to generate domain models [20]. It is based on an extension of first order logic. A PDDL domain model contains a set of actions that represent operators. In early days of AP, STRIPS was the most popular representation language. Later PDDL was developed for the first international planning conference (IPC) and since that date it has become a standard language for the AI planning community [21].

ii) What inputs are there to the learning process?

By input to the system we mean the feed based on which the system generates a partial or full domain model. Input to the learning process could be training examples, observations, constraints, initial and goal states, predicates and in some systems a partial domain model to generate a full domain model. The expressiveness of the learned domain model is directly proportional to the demand of more detailed input [21]. Some recent work on learning domain models has focused on learning with little or no supplied domain knowledge and LOCM is a recent system that learns from plan scripts only [4].

iii) What level of learning is taking place?

The learning outcome is normally the acquisition of a set of operator schema or methods, but could also include a set of heuristics to be used to speed up planning. Output is a PDDL domain model. A few systems generate an initial level of domain model at the first attempt that later could be refined and enhanced while some of them produce solver-ready domain models as well.



Block Diagram of a typical Learning System

Existing Automated tools for Domain Model generation

There are three main categories of approaches based on the phase of planning that learning is applied to and the consequent type of knowledge that is acquired. It includes Domain Knowledge, Control Knowledge and Optimization Knowledge. Only Domain Knowledge lies in the scope of this paper. **Domain knowledge** is utilized by planners in pre-processing phases in order to either modify the description of the problem in a way that will make it easier for solving it or make the appropriate adjustments to the planner to best attack the problem [7, 11].

Based on the different types of inputs systems take and the level of domain model they generate, there are several different automated systems/algorithms developed by different authors e.g. ARMS, LAMP, Opmaker, RIMS, SLAF, LOCM and several others. All these systems fall in the domain knowledge category and work on their own developed methodology. These are briefly discussed here:

ARMS *(Action-Relation Modelling System)*

ARMS [10] is a tool for learning action schema from observed plans with partial information. It is a system for automatically discovering action models from a set of observed plans where the intermediates states are either unknown or only partially known. To learn action schema, ARMS gathers knowledge on the statistical distribution of frequent sets of actions in the example plans. It then forms a weighted propositional satisfiability (Weighted SAT) problem and resolves it using a weighted MAX-SAT solver.

ARMS operates in two phases, where it first applies a frequent set mining algorithm to find the frequent subsets of plans that need be explained first and then applies a SAT algorithm for finding a consistent assignment of preconditions and effects [10].

Opmaker

Developed by Richardson [11], Opmaker is a method for inducing primitive and hierarchical actions from several examples, without the use of intermediate states as in ARMS. In Opmaker, actions are generated from relatively short action sequences. The user simply has to name an action and identify associated objects as being affected or unaffected by the action. The system uses the static domain knowledge, the initial and goal states and planning sequence as input. Using these it first deduces possible state-change pathways and uses these to induce the actions it needs [11]. These actions can then be learned or regenerated and improved according to requirement.

The real motivation for writing Opmaker algorithm came from the need to extend GIPO, an integrated package for the construction of domain models in the form of graphical user interface [4].

Opmaker 2

Opmaker2 (an extension of Opmaker) is a knowledge acquisition and formulation tool, which inputs a domain knowledge and a training sequence, and outputs a set of PDDL operator schema. It followed on from the original Opmaker idea [11]. Its aims are similar to systems such as ARMS [10] in that it

supports the automated acquisition of a set of operator schema that can be used as input to an automated planning engine.

Both systems have their own methodology. In ARMS it inputs many training examples and constraints, and is based on a propositional-style (PDDL) representation while Opmaker2 requires only one example of each operator schema that it learns in addition to partial domain model as input [12].

RIMS *(Refining Incomplete Planning Domain Models)*

RIMS [13] is a system designed for situations where a planning agent has an incomplete model which it needs to refine through learning. This method takes as input an incomplete model (with missing preconditions and effects in the actions), and a set of plans that are known to be correct. It outputs a "refined" model that not only captures additional precondition/effect knowledge about the given actions, but also "macro actions". A macro-action can be defined as a set of actions applied at single time, can quickly reach a goal at less depth in the search tree and thus problems which take a long time to solve, might become solvable quickly [22].

In the first phase, it looks for candidate macros found from the plan traces, and in the second phase it learns precondition/effect models both for the primitive actions and the macro actions. Finally it uses the refined model to do planning [13]. The running time of this system increases polynomially with the number of input plan traces.

LAMP *(Lazy Adaptive Multi-criteria Planning)*

The purpose of the Lazy Adaptive Multi-criteria Planning (*LAMP*) system is to configure a planner in an optimal way. Its major concern is about two quality criteria i.e. to improve the execution speed of planning systems and the quality of their solutions for a given problem according to user-specified preferences [23].

The LAMP system has some advantages over other compared systems, in that it allows the users to specify their choice regarding plan length and planning time. It can also incrementally learn by running the planner off-line when new problems arise. LAMP system is quite faster in learning due to its lazy nature and only fractionally slower at classification time, and experimental results show that LAMP improves the performance of the default configuration of two already well-performing planning systems in a variety of planning problems.

SLAF

Shahaf and Amir presented an algorithm that generates actions‟ effects and preconditions in partially observable domains. Preconditions and effects that this system generates include implicit objects and unspecified relationships between objects through the use of action schema language. It can also learn models with conditional effects. After partial observations and a sequence of executed actions the good thing about this algorithm is that it keeps and outputs a relational logical picture of all possible action-schema models and this relational picture not only fasten up learning process and computation but also allows generalization from objects to classes [14].

This algorithm takes advantage of DAG (Direct Acyclic Graph) representation of the formula. The results from this algorithm can be used in deterministic domains which involve many actions, relations and objects.

LOCM *(Learning Object Centred Models)*

LOCM is distinct from other systems that learn action schema from examples in that it requires only the action sequences as input to produce the required action schema as output. Its success is based on the assumption that the output domain model can be represented in an object centred representation [18]. Although LOCM requires no background information, it requires many instances of plans before it can synthesize domain models [24].

LOCM uses assumptions about the action schema it has to produce. The first assumption is about the object set called „sort", where the behaviour of each object in a single sort is similar. On action execution all like objects in one sort can change certain specific states only. Another assumption is that LOCM makes observations about domain knowledge, and these observations are sequences of possible action applications. Using these assumptions, LOCM can construct an action model without the need for extra input information. This system has been implemented in Prolog incorporating its own algorithm.

CONCLUSIONS

The technical review of different domain learning systems and methodologies reveals that a typical domain learning system consists of different components to accept plan traces or sometimes additional information from a user and deliver full or partial domain model back to the user. These components are used to *learn* from the input information (training data) user pass and through sequential and logical arrangement of actions in plans, they process plans through designed learning methodology of system and generates a PDDL domain model which consists of action schema/operators. A few systems generate an initial level of domain model at the first attempt that later could be refined and enhanced while some of them produce solver-ready domain models as well.

Input to the learning process could be training plans, observations, constraints, initial and goal states, predicates and in some systems partial domain model to generate a full domain model. The correctness and accuracy of the learned domain model increases with the increase in the demand input assistance. Latest work on such systems makes in enable to work only with multiple instances of plans as input.

The working of mentioned systems is beneficial in domains where automatic logging of some existing process can produce enough training data, e.g. games, workflow, or online transactions. As a future work, in addition to enhancement in automated knowledge engineering tools, a tool is required which can generate training data to be used as input to these systems, for domains that do not currently have training sequences available.

REFERENCES

1. Shah, M., et al., *Knowledge Engineering Tools in Planning: State-of-the-art and Future Challenges.* Knowledge Engineering for Planning and Scheduling: p. 53.
2. Elementary, E.i. *Rule-based Knowledge (Expert) Systems*. Available from: www.eie.polyu.edu.hk/~enzheru/.../eie426-knowledge-systems-v3.pp.
3. Vaquero, T., et al. *itSIMPLE4. 0: Enhancing the modeling experience of planning problems*. in *Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS-12)–System Demonstration*. 2012.
4. Simpson, R.M., D.E. Kitchin, and T. McCluskey, *Planning domain definition using GIPO.* Knowledge Engineering Review, 2007. **22**(2): p. 117-134.
5. González-Ferrer, A., J. Fernández-Olivares, and L. Castillo, *JABBAH: a Java application framework for the translation between business process models and HTN.* Proceedings of the 3rd International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS'09), 2009.
6. Barreiro, J.e.a. *EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization.* in *In Proceedings of the 22nd International Conferenceon Automated Planning & Scheduling (ICAPS-12) – The 4th International Competition on Knowledge Engineering for Planning and Scheduling.* 2012.
7. Bouillet, E.e.a. *Mario: middleware for assembly and deployment of multi-platform flow-based applications.* in *In Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware, Middleware '09, 26:1–26:7. New York, NY, USA: Springer-Verlag NewYork, Inc.* 2009.
8. Vodrázka, J. and L. Chrpa. *Visual design of planning domains*. in *Proceedings of ICAPS 2010 workshop on Scheduling and Knowledge Engineering for Planning and Scheduling (KEPS)*. 2010.
9. Plch, T., et al., *Inspect, Edit and Debug PDDL Documents: Simply and Efficiently with PDDL Studio.* and Exhibits, 2012: p. 15.
10. Wu, K., Q. Yang, and Y. Jiang, *Arms: Action-relation modelling system for learning action models.* CKE, 2005: p. 50.
11. Richardson, N.E., *An operator induction tool supporting knowledge engineering in planning*, 2008, University of Huddersfield.
12. McCluskey, T., et al., *An evaluation of Opmaker2.* 2008.
13. Zhuoa, H.H., T. Nguyenb, and S. Kambhampatib. *Refining incomplete planning domain models through plan traces*. in *Proceedings of IJCAI*. 2013.
14. Shahaf, D. and E. Amir. *Learning partially observable action schemas*. in *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*. 2006. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.
15. Cresswell, S., *LOCM: A tool for acquiring planning domain models from action traces.* 2009.
16. Cresswell, S., T.L. McCluskey, and M.M. West. *Acquisition of Object-Centred Domain Models from Planning Examples*. in *ICAPS*. 2009.
17. Vaquero, T.S., J.R. Silva, and J.C. Beck, *A brief review of tools and methods for knowledge engineering for planning & scheduling.* KEPS 2011, 2011: p. 7.
18. Cresswell, S.N., T.L. McCluskey, and M.M. West, *Acquiring planning domain models using LOCM.* Knowledge Engineering Review, 2013.
19. al, L.C.e., *Machine learning and adaptation to domain models to support real time planning in autonomous system.*
20. Fox, M. and D. Long, *Modelling Mixed Discrete-Continuous Domains for Planning.* J. Artif. Intell. Res.(JAIR), 2006. **27**: p. 235-297.
21. Fox, M. and D. Long, *PDDL2. 1: An Extension to PDDL for Expressing Temporal Planning Domains.* J. Artif. Intell. Res.(JAIR), 2003. **20**: p. 61-124.
22. Newton, M., J. Levine, and M. Fox, *Genetically evolved macro-actions in AI planning problems.* Proceedings of the 24th UK Planning and Scheduling SIG, 2005: p. 163-172.
23. Tsoumakas, G., et al. *Lazy adaptive multicriteria planning*. in *ECAI*. 2004.
24. Shah, M.M.S., *An Investigation into Using Object Constraints to Synthesize Planning Domain*, in *ICAPS 2009 Doctoral Consortium*2009: Thessaloniki, Greece. p. 53-56.