Automated Knowledge Engineering Tools in Planning: State-of-the-art and Future Challenges

Rabia Jilani and **Andrew Crampton** and **Diane Kitchin** and **Mauro Vallati**
School of Computing and Engineering
University of Huddersfield
United Kingdom

Abstract

Intelligent agents must have a model of the dynamics of the domain in which they act. Models can be encoded by human experts or, as required by autonomous systems, automatically acquired from observation. At the state of the art, there exist several systems for automated acquisition of planning domain models.

In this paper we present a brief overview of the automated tools that can be exploited to induce planning domain models. While reviewing the literature on the existing tools for Knowledge Engineering (KE), we do a comparative analysis of them. The analysis is based on a set of criteria. The aim of the analysis is to give insights into the strengths and weaknesses of the considered systems, and to provide input for new, forthcoming research on KE tools in order to address future challenges in the automated KE area.

Introduction

Both knowledge acquisition and knowledge engineering for AI planning systems are essential to improve their effectiveness and to expand the application focus in practice. The improvement process includes the study of planning application requirements, creating a model that explains the domain, and testing it with suitable planning engines to get a final product which consists of a domain model. Domain models can be encoded by human experts or automatically learned through the observation of some existing plans (behaviours). Encoding a domain model from observations is a very complex and time-consuming task, even for domain experts. Various approaches have been used to learn domain models from plans. This is of increasing importance: domain independent planners are now being used in a wide range of applications, but they should be able to refine their knowledge of the world in order to be exploited also in autonomous systems. Automated planners require action models described using languages such as the Planning Domain Definition Language (PDDL) (Mcdermott et al. 1998).

There have been reviews of existing knowledge engineering tools and techniques for AI Planning (Vaquero, Silva, and Beck 2011; Shah et al. 2013). Vaquero et al. (2011) provided a review of tools and methods that address the challenges encountered in each phase of a design process. Their work covers all the steps of the design cycles, and is focused on tools that can be exploited by human experts for encoding domain models. Shah et al. (2013) explored the deployment of automated planning to assist in the development of domain models for different real-world applications.

Currently, there is no published comparison research on KE tools for AI planning that automatically encode a domain model from observing plan traces. In this paper, we compare and analyse different state-of-the-art, automated KE tools that automatically discover action models from a set of successfully observed plans. Our special focus is to analyse the design issues of automated KE systems, the extent of learning that can take place, the inputs that systems require and the competency in the output domain model which systems induce for dealing with complex real problems. We evaluate nine different KE tools against the following criteria: (i) Input Requirements (ii) Provided Output (iii) Language (iv) Noise in Plans (v) Refinement (vi) Operational Efficiency (vii) User Experience, and (viii) Availability. By evaluating state-of-the-art tools we can gain insight into the quality and efficiency of systems for encoding domain models, and better understand the improvements needed in the design of future supporting tools.

The rest of this paper is organised as follows. We first provide an overview of existing automated KE tools for supporting the task of encoding planning domain models. Then we discuss the criteria that are used for comparing the different encoding methods. Finally, we summarise some guidelines for future tools.

The State of the Art

In this section we provide an overview of KE tools that can be used for automatically producing planning domain models from existing plans or sequences of actions.

Opmaker

Opmaker (McCluskey, Richardson, and Simpson 2002) is a method for inducing primitive and hierarchical actions from examples, in order to reduce the human time needed for describing low level details related to operators' pre- and post-conditions.

Opmaker is an algorithm for inducing parameterized, hierarchical operator descriptions from example sequences and declarative domain knowledge (object hierarchy, object descriptions, etc.)

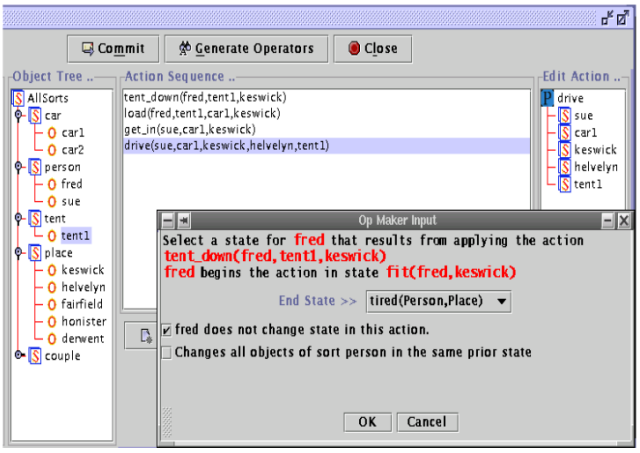

Figure 1: A screen shot of Opmaker.

The user has to specify an action and identify associated objects as being affected or unaffected by the action. The system uses static domain knowledge, the initial and goal states and a planning sequence as input. Using this knowledge, it first deduces possible state-change pathways and then uses them to induce the needed actions. These actions can then be learned or regenerated and improved according to requirement.

Opmaker extends GIPO, an integrated package for the construction of domain models, using a graphical user interface (Simpson, Kitchin, and McCluskey 2007). Figure 1 shows a screen shot of the graphical user interface.

SLAF

The SLAF (Simultaneous Learning and Filtering) algorithm (Shahaf and Amir 2006) learns action models in partially observable domains. As inputs, SLAF includes specifications of fluents, as well as partial observations of intermediate states between action executions. The pre-conditions and effects that this system generates in output includes implicit objects and unspecified relationships between objects through the use of action schema language.
As output the system learns action models (pre-conditions and effects) that also include conditional effects through a sequence of executed actions and partial observations. The action schema from this algorithm can be used in deterministic domains which involve many actions, relations and objects. This algorithm uses a 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.

ARMS

ARMS (Action-Relation Modelling System) (Yang, Wu, and Jiang 2007) 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 intermediate 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 share a common set of parameters. It then applies a SAT algorithm for finding a consistent assignment of preconditions and effects.

ARMS needs partial intermediate states in addition to observed plan traces as input. The action model learnt from ARMS is not guaranteed to be completely correct, as the domain model induced is based on guesses with a minimal logical action model. This is why it can only serve as an additional component for the knowledge editors which provide advice for human users, such as GIPO (Simpson, Kitchin, and McCluskey 2007), and not as an independent, autonomous agent.

Opmaker2

Opmaker2 (an extension of Opmaker) (McCluskey et al. 2009) is a knowledge acquisition and formulation tool, which inputs a partial domain model and a training sequence, and outputs a set of PDDL operator schema including heuristics that can be used to make plan generation more efficient. It follows on from the original Opmaker idea. Its aims are similar to systems such as ARMS in that it supports the automated acquisition of a set of operator schema that can be used as input to an automated planning engine. Opmaker2 determines its own intermediate states of objects by tracking the changing states of each object in a training example sequence and making use of partial domain knowledge provided with input. Opmaker2 calls it the DetermineState procedure. The output from DetermineState is a map of states for each object in the example sequence. Parameterized operator schema are generated after applying the Opmaker algorithm for generalization of object references collected from example sequences.

LOCM

LOCM (Learning Object Centred Models) (Cresswell, McCluskey, and West 2013) is significantly different from other systems that learn action schema from examples. It requires only a set of valid plans as input to produce the required action schema as output. Valid plans should be formatted in a specific way; an example is given in Figure 2. LOCM is based on the assumption that the output domain model can be represented in an object-centred representation (Cresswell, McCluskey, and West 2013). Using an object-centred representation, LOCM outputs a set of parameterized Finite State Machines (FSMs) where each FSM represents the behaviour of each object in the learnt action schema. Such FSMs are then exploited in order to identify pre- and post-conditions of the domain operators. Although LOCM requires no background information, it usually requires many plan traces for synthesizing meaningful domain models. Moreover, LOCM is not able to automatically identify and encode static predicates.

```
sequence_task(1, [unstack(b8, b9),
stack(b8, b10), pick-up(b7), stack(b7,
b8), unstack(b9, b1), put-down(b9),
unstack(b1, b3), stack(b1, b9),
unstack(b3, b2), stack(b3, b6),
pick-up(b5), stack(b5, b3), unstack(b7,
b8), stack(b7, b2), unstack(b8, b10),
stack(b8, b7), pick-up(b10), stack(b10,
b5)], _, _).
```

Figure 2: An example of a blocksworld plan formatted as required by LOCM.

LOCM2

LOCM2 (Learning Object Centred Models 2) (Cresswell and Gregory 2011) followed on from the LOCM idea. Experiments have revealed that there are many examples that have no model in the representation used by LOCM. A common feature of domains which produce this issue is one where objects can have multiple aspects of their behaviour, and so they need multiple FSMs to represent each object's behaviour. LOCM2 generalizes the domain induction of LOCM by allowing multiple parameterised state machines to represent a single object, with each FSM characterised by a set of transitions. This enables a varied range of domain models to be fully learned. LOCM2 uses a transition-centred representation instead of the state-centred representation used by LOCM. The current LOCM and LOCM2 systems gather only the dynamic properties of a planning domain and not the static information. While domains used in planning also depend on static information, research is being carried out to fill that gap and make these systems able to induce both the dynamic and the static parts of domain models.

LSO-NIO

The system LSO-NIO (Learning STRIPS Operators from Noisy and Incomplete Observations) (Mourão et al. 2012) has been designed for allowing an autonomous agent to acquire domain models from its raw experience in the real world. In such environments, the agent's observation can be noisy (incorrect actions) and incomplete (missing actions). In order to acquire a complete STRIPS (Fikes and Nilsson 1972) domain model, the system requires a partial model, which describes objects' attributes and relations, and operators' names.

LSO-NIO exploits a two-staged approach. As a first stage, LSO-NIO learns action models by constructing a set of kernel classifiers, which are able to deal with noise and partial observability. The resulting models are "implicit" in the learnt parameters of the classifiers (Mourão, Petrick, and Steedman 2010). The implicit models act as a noise-free and fully observable source of information for the subsequent step, in which explicit action rules are extracted. The final output of LSO-NIO is a STRIPS domain model, ready to use for domain-independent planners.

RIM

RIM (Refining Incomplete Planning Domain Models) (Zhuo, Nguyen, and Kambhampati 2013) 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 pre-conditions and effects in the actions), and a set of plans that are known to be correct. By executing given plan traces and preconditions/effects of the given incomplete model, it develops constraints and uses a MAX-SAT framework for learning the domain model (Zhuo et al. 2010). 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 a single time, that 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.

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 plan. The running time of this system increases polynomially with the number of input plan traces.

In the RIM paper the authors provide a comparison between RIM and ARMS by solving 50 different planning problems; through action models, refined and induced by the two systems. RIM uses both plans and incomplete domain models to induce a complete domain model but ARMS uses plans only, so to keep both systems' output on the same scale, RIM induces action models (used for comparison) based on plan traces only.

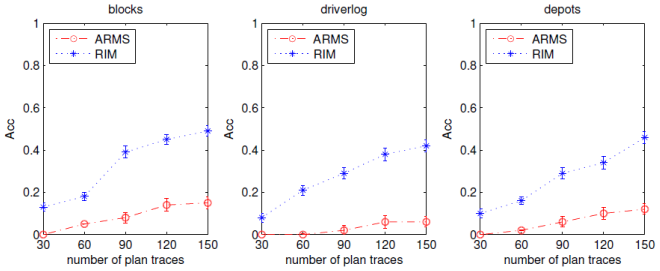

Figure 3: Comparison between RIM and ARMS (Zhuo, Nguyen, and Kambhampati 2013).

The average length of a plan is 18 when using action models learnt by ARMS; while the average length of plans (to the same problems as solved by ARMS) is 21. This is when using preferences of macro-operators learnt by RIM. Figure 3 shows a comparison of three different domains; the correctness of RIM is better than ARMS, as RIM also learns macro-operators and it uses macros to increase the accuracy of plans generated with the refined system.

AMAN

AMAN (Action-Model Acquisition from Noisy plan traces) (Zhuo and Kambhampati 2013) was designed to create domain models in situations where there is little or no possibil-

ity of collecting correct training data (plans). Usually, noisy plan traces are easier and cheaper to collect. An action is considered to be noisy if it is mistakenly observed.

AMAN works as follows. It builds a graphical model to capture the relations between actions (in plan traces) and states, and then learns the parameters of the graphical model. After that, AMAN generates a set of action models according to the learnt parameters. Specifically, AMAN first exploits the observed noisy plan traces to predict correct plan traces and the domain model based on the graphical model, and then executes the correct plan traces to calculate the reward of the predicted correct plan traces according to a predefined reward function. Then, AMAN updates the predicted plan traces and domain model based on the reward. It iteratively performs the above-mentioned steps until a given number of iterations is reached. Finally, the predicted domain model is provided.

In the AMAN paper, a comparison of AMAN and ARMS (Yang, Wu, and Jiang 2007) on noiseless inputs is provided.

Criteria for Evaluating Tools

We have identified several criteria that are useful for evaluating the existing KE automated tools for inducing domain models. Such criteria have been designed for investigating the KE tools' functionality from different perspectives: input, output, efficiency, availability and usability.

Input Requirements:

What inputs are required by a system to refine/induce a partial or full domain model? Input to the learning process could be training plans, observations, constraints, initial and goal states, predicates, and in some systems a partial domain model (with missing pre-conditions and effects in the actions).

Provided Output:

What is the extent of learning that the system can do?

Language:

What language does the system support to produce the output domain model? e.g. PDDL, STRIPS, and OCL etc.

Noise in Plans:

Is the tool able to deal with noise in plans? Noise in plans can be either incomplete plan traces (i.e., missing actions) or noisy actions. An action in a plan is considered to be noisy if it is incorrectly observed.

Refinement:

Does the tool refine existing domain models or does it build domain models from scratch?

Operational Efficiency:

How efficiently are the models produced? In general terms, the efficiency of a system could be seen as the ratio between input given to the system to do the learning process and the output domain model that we get as a result of learning.

User Experience:

Is the system/tool designed for inexperienced/beginner level planning users? Do users need to have a good knowledge of the system output language?

Availability and Usage:

Is the system available for open use? Does the system provide a user manual?

Tools Evaluation

In this section all the KE tools introduced in this paper are evaluated against the outlined criteria. Table 1 shows an overview of the comparison.

Inputs Requirements

The input to RIM, LOCM, LOCM2 and ARMS is a correct sequence of actions (training data in the form of plan traces), where each action in a plan is stated as a name and a list of objects that the action refers to. For some domains which require static knowledge, there is a need to mention static pre-conditions for the domain to be learnt; as LOCM and LOCM2 cannot learn static aspects of the domain. RIM in addition to a correct action sequence also requires an incomplete domain model (with missing pre-conditions and effects in the actions) as an input. ARMS makes use of background knowledge as input, comprising types, relations and initial and goal states to learn the domain model.

In comparison Opmaker2 learns from a single, valid example plan but also requires a partial domain model (declarative knowledge of objects hierarchy, descriptions, etc) as input.

AMAN and LSO-NIO, these systems learn from noisy (incorrect actions) and incomplete (missing actions) observations of real-world domains. Just like Opmaker2, LSO-NIO also requires a partial domain model, which describes objects (and their attributes and relations) as well as the name of the operators. The inputs to SLAF include specifications of fluents, as well as partial observations of intermediate states between action executions.

Provided Output

ARMS, Opmaker2, LOCM, LOCM2, LSO-NIO and RIM, the output of these systems is a complete domain model. In addition, LOCM also displays a graphical view of the finite state machines, based on which the object behaviour in the output model is learnt. To increase the efficiency of plans generated, Opmaker2 also includes heuristics while RIM also learns macro operators.

SLAF, as output the system learns an action model (pre-conditions and effects) that also includes conditional effects through a sequence of executed actions and partial observations.

Given a set of noisy action plans, AMAN generates multiple (candidate) domain models. To capture domain physics it produces a graphical model and learns its parameters.

Language

The domain model (also called domain description or action model) is the description of the objects, structure, states, goals and dynamics of the domain of planning (McCluskey et al. 2009).

LOCM, LOCM2 and ARMS are able to provide a PDDL domain model representation. RIM, AMAN and LSO-NIO can handle the STRIPS subset of PDDL. Opmaker and Opmaker2 use a higher level language called Object Centred Language (OCL) (McCluskey, Liu, and Simpson 2003; McCluskey and Porteous 1997) for domain modelling. Their output is an OCL domain model, but Opmaker can exploit the GIPO tool to translate the generated models into PDDL. Finally, SLAF System is able to exploit several languages to represent action schemas; starting from the most basic language SL, and then there is SL-V and SL-H (Shahaf and Amir 2006). Such languages are not usually supported by domain-independent planners.

Noise in Plans

Most of the existing KE tools require valid plans. AMAN and LSO-NIO are the systems that can deal with noisy plan traces. Moreover, LSO-NIO is also able to handle incomplete plan traces. On the other hand, also LAMP (Zhuo et al. 2010), on which RIM is based, is able to exploit partial plan traces, in which some actions are missing.

Refinement

Most existing work on learning planning models learns a complete new domain model. The only tool among all those reviewed in the paper that is able to refine an existing domain model is RIM. RIM takes in correct plan traces as well as an incomplete domain model (with missing pre-conditions and effects in the actions), to refine it by capturing required pre-condition/effects.

On the other hand, since Opmaker, Opmaker2, LSO-NIO and SLAF require as input part of the domain knowledge, to some extent they are actually refining the provided knowledge.

Operational Efficiency

The efficiency of a system could be seen as the ratio between the input given to the system for the learning process and the output domain model we get as a result of learning. As shown in the review of the considered tools, all systems have different and useful motivations behind their development. Given their relevant features of input requirements and learning extent, we can say that the system needing the least input assistance and which induces the most complete domain model is the most efficient one. On such a scale the AMAN, LOCM and LOCM2 approaches have the best performance. In order to provide a complete domain model they require only some plan traces (sequence of actions). Based on strong assumptions they output the solver-ready domain model. Similarly ARMS, though requiring richer inputs, outputs a solution which is optimal; in that it checks error and redundancy rates in the domain model and reduces them. In contrast Opmaker2 learns from a single example together with a partial domain model, and for output it not only produces a domain model, but also includes heuristics that can be used to make plan generation through the domain model more efficient.

User Experience

By experience we mean to evaluate how far the system/tool is designed for use by inexperienced/beginner level planning users. Most of these systems are built with the motivation to open up planning engines to general use. Opmaker is incorporated into GIPO as an action induction system, as GIPO is an integrated package for the construction of domain models in the form of a graphical user interface. It is used both as a research platform and in education. It has been used to support the teaching of artificial intelligence (AI) planning to students with a low-experience level (Simpson, Kitchin, and McCluskey 2007).

The other systems are being used as standalone systems, they do not provide a GUI, and require the guidance of planning experts for usage. Certain systems also require separate formats for providing inputs, e.g., LOCM requires input plan traces in Prolog while many major planning engines use PDDL as a planning language. So the conversion from PDDL to Prolog is a time consuming task and requires experienced users.

Availability and Usage

Very few systems are available on-line and open to download and practice. No systems provide documentation that make usage easy for beginners - except GIPO (Opmaker).

Guidelines and Recommendations

We will now discuss the guidelines and recommendations that we have derived from our review and assessment of the nine different state-of-the-art automated KE tools.

To create or refine an already existing domain model requires many plan examples and sometimes other inputs such as full or partial knowledge about predicates, initial, intermediate and goal states, and sometimes a partial domain model. One major concern at this stage is the way plan traces can be collected. There are three general ways to collect example plans. The first is when plans are generated through goal oriented solutions, the second through random walks and thirdly through observation of the environment by a human or by an agent. Goal oriented plan solutions are generally expensive in that a tool or a planner is needed to generate a large enough number of correct plans to be used by the system. To do this one must also have a pre-existing domain model. Observation by an agent has a high chance that noise will be introduced in the plan collection; which can clearly affect the learning process. Currently most working systems assume the input knowledge to be correct and consequently not suitable for real-world applications. To increase potential utility, systems should be able to show equal robustness to noise.

Another issue is the expressiveness of the output domain model. Observing the output of the current automated learning systems, there is a need to extend their development so that they can also learn metric domains that include durative actions, action costs and other resources. In other words, the

Criteria	AMAN	ARMS	LOCM	LOCM2	LSO-NIO	Opmaker	Opmaker2	RIM	SLAF
Inputs	NP	BK,P	P	P	PDM,NP	PDM,P	PDM,P	PDM,P	Pr,IS
Outputs	DM	DM	DM	DM	DM	DM	DM,H	RDM	DM
Language	STRIPS	PDDL	PDDL	PDDL	STRIPS	OCL	OCL	STRIPS	SL
Noise	$+$	$-$	$-$	$-$	$+$	$-$	$-$	$-$	$-$
Refinement	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$+$	$-$
Efficiency	$+$	i	$+$	$+$	i	$-$	i	$-$	$-$
Experience	$-$	$-$	$-$	$-$	$-$	$+$	$+$	$-$	$-$
Availability	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$-$	$-$

Table 1: Comparison of KE Tools. P: Plan traces; BK: Background Knowledge; PDM: Partial Domain Model; Pr: Predicates; IS: Intermediate States; NP: Noisy Plans; DM: Domain Model; RDM: Refined Domain Model; H: Heuristics. Where available, $+$, i (intermediate) or $-$ give a qualitative evaluation w.r.t. the corresponding metric.

systems should broaden the scope of domain model generation to produce more expressive versions of PDDL that can be applied to a greater range of real-world problems.
Systems which learn only from plan traces could make the output domain model more intelligible and useful by assigning meaningful names to all learnt fluents/predicates.
To enhance the potential utility of the induced domain in the real-world, error and redundancy checks should be performed in order to enhance the effectiveness of plans generated by planning engines using these domains.
To make learning systems more accessible and open to use by research students and the scientific community, these systems should be available on-line, and include a GUI and user manual for ease of use by non-planning experts. A significant extension would be to create a consistent interface across all systems for specifying inputs. Having to convert PDDL plans into Prolog, for example, is likely to inhibit the uptake of automated KE tools by non-experts rather than encourage it.

Conclusion

In order to encourage the exploitation of Automated Planning in autonomous systems, techniques for the automatic acquisition of domain models are of fundamental importance. Providing robust and expressive automated KE tools for domain model acquisition, that can be easily used by non-planning experts, will better promote the application of planning in real-world environments; particularly in applications where the actual domain model is unclear and/or too complex to design manually.

In this paper we have presented the state-of-the-art of Knowledge Engineering tools for the automatic acquisition of planning domain models. We proposed and used a set of criteria consisting of: input requirements, output, efficiency, supported language, ability to handle noisy plans, ability to refine existing models, user experience and availability. We observed that different tools require very different inputs and, usually, are designed for experienced users. We highlighted the weaknesses of existing methods and tools and we discussed the need for PDDL-inspired development in the design of future tool support.

Future work will include an experimental comparison, based on a case-study. We are also interested in improving existing KE tools for overcoming the major weaknesses that this review has highlighted.

References

Cresswell, S., and Gregory, P. 2011. Generalised domain model acquisition from action traces. In *Proceedings of The 21th International Conference on Automated Planning & Scheduling (ICAPS-11)*.

Cresswell, S. N.; McCluskey, T. L.; and West, M. M. 2013. Acquiring planning domain models using locm. *The Knowledge Engineering Review* 28(02):195–213.

Fikes, R. E., and Nilsson, N. J. 1972. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2(3):189–208.

McCluskey, T. L., and Porteous, J. M. 1997. Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence* 95(1):1–65.

McCluskey, T. L.; Cresswell, S.; Richardson, N. E.; and West, M. M. 2009. Automated acquisition of action knowledge.

McCluskey, T.; Liu, D.; and Simpson, R. M. 2003. Gipo ii: Htn planning in a tool-supported knowledge engineering environment. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003)*, volume 3, 92–101.

McCluskey, T. L.; Richardson, N. E.; and Simpson, R. M. 2002. An interactive method for inducing operator descriptions. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems*, 121–130.

Mcdermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Mourão, K.; Zettlemoyer, L. S.; Mark, R. P.; and Steedman. 2012. Learning strips operators from noisy and incomplete observations. In *Proceedings of the Twenty Eighth Conference on Uncertainty in Artificial Intelligence (UAI-12)*, 614–623.

Mourão, K.; Petrick, R. P. A.; and Steedman, M. 2010. Learning action effects in partially observable domains. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI-10)*, 973–974.

Shah, M.; Chrpa, L.; Jimoh, F.; Kitchin, D.; McCluskey, T.; Parkinson, S.; and Vallati, M. 2013. Knowledge engineering tools in planning: State-of-the-art and future challenges. *Knowledge Engineering for Planning and Scheduling* 53.

Shahaf, D., and Amir, E. 2006. Learning partially observable action schemas. In *The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference (AAAI-06)*, volume 21, 913. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. 2007. Planning domain definition using gipo. *The Knowledge Engineering Review* 22(02):117–134.

Vaquero, T. S.; Silva, J. R.; and Beck, J. C. 2011. A brief review of tools and methods for knowledge engineering for planning & scheduling. In *Proceedings of the Knowledge Engineering for Planning and Scheduling workshop – The 21th International Conference on Automated Planning & Scheduling (ICAPS-11)*.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence* 171(2-3):107–143.

Zhuo, H. H., and Kambhampati, S. 2013. Action-model acquisition from noisy plan traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI-13)*, 2444–2450. AAAI Press.

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.

Zhuo, H. H.; Nguyen, T.; and Kambhampati, S. 2013. Refining incomplete planning domain models through plan traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, 2451–2457. AAAI Press.