Towards Model-lite Planning: A Proposal For Learning & Planning with Incomplete Domain Models

Sungwook Yoon
Computer Science & Engineering
Arizona State University
Tempe, AZ 85281
Sungwook.Yoon@asu.edu

Subbarao Kambhampati
Computer Science & Engineering
Arizona State University
Tempe, AZ 85281
rao@asu.edu

Abstract

Due to the difficulty of defining a *complete* domain definition, it is hard to apply planners to real-life application domains. *Model-lite* planning research has been proposed to address this issue (Kambhampati 2007). While it takes time to develop a complete domain definition, it is rather faster and easier to provide a planner with incomplete but close to complete domain definition. In this paper, we describe our on-going research on a new learning and planning scheme for the model-lite paradigm. In our framework, we use *probabilistic logic* to leverage the incomplete model and we consider explicit input background knowledge. In the planning side, the core idea is viewing a planning problem as an MPE (most plausible explanation) problem, given incomplete domain definition and the evidence of initial state and goal. We show a reduction procedure from a planning problem to an MPE problem then to a MAXSAT problem, which enables us to use MAXSAT solvers, as deterministic planning uses SAT solvers. In the learning side, we show how we update the domain model, given trajectories using a weighted logic learning package, and how we parsimoniously generate axioms with background knowledge. This is particularly important to leverage the relational structure during planning.

Introduction

There are many application areas where planning techniques can play an important role. Web service composition, automated computing or dynamic compilation are some of them (Kambhampati 2007). One of the reasons why planning techniques are not being actively used in these areas is the burden of developing domain definition. It is well understood among the planning community that defining a complete domain for the target application is not easy (Bartk & McCluskey 2005; Yang, Wu, & Jiang 2007). Indeed, DARPA has started the Integrated Learning Initiative to address this issue (DARPA 2006). Recently, (Kambhampati 2007) has proposed the model-lite planning paradigm, which motivates the study of planning techniques that do not rely on the complete domain definition.

Although it is hard to write a complete domain definition, it may be relatively easy to provide an approximate domain definition which may not be complete. The challenge is to plan with the current-best model and update it as experiences are gathered. In this paper, we describe our on-going research on the learning and planning framework with incomplete domain models. We use probabilistic logic to represent uncertain domain models. We separate precondition and effect of action definitions into two different sets of axioms, precondition and effect axioms, where each of the axiom has an associated probability. This separation helps a straightforward setup of a well known machine learning problem. Then, we show that planning can be viewed as an MPE (most probable explanation) problem, given the initial state, goal and the current domain model. Next, we view MPE problem as weighted MAXSAT problem, as considered by (Park 2002). Thus, the planning problem with an incomplete domain model becomes a weighted MAXSAT problem, which enables us to use publicly available approximate solvers that scales well, e.g., maxwalksat. Note that our proposed approach for planning can also be applied to probabilistic planning.

There are two aspects of learning in our framework.

- Given a set of weighted precondition and effect axioms, the learning is mainly updating their weight. A weight update on logic formulae can straightforwardly be achieved with MLN (Markov logic network) (Richardson & Domingos 2006) or Problog (De Raedt, Kimmig, & Toivonen 2007) approaches. We show how we construct training data and how the learning updates the domain model.
- Given explicit background knowledge, we show how we use the knowledge to reduce the number of potential candidate precondition or effect axioms. This will make later planning easier, as the plangraph construction will be lighter.

Explicit consideration of background knowledge is an interesting point of our framework. At first glance, it might be surprising that we assume incomplete domain model but expect background knowledge. However, this is not an unreasonable assumption. In most domain modeling tasks, the problem is not so much that no background knowledge is available, but that it is hard to come up with complete and consistent set of operators that respect all the background knowledge. This is particularly useful when there are obvious domain constraints, e.g. no more than one block on top of a block in Blocksworld, but it is not easy to create a set of

actions that maintains the property.

Representation

A planning domain $\mathbb{D}$ specifies a tuple $\{\mathbb{P}, \mathbb{A}, Pre, Eff, \mathbb{B}\}$. $\mathbb{P}$ defines available predicates and their arities. $\mathbb{A}$ defines available action templates and arities of them. Pre is a set of precondition axioms. For each action type $a \in \mathbb{A}$, there is at least one precondition axiom $P \in Pre$, s.t. $(w_p, a \rightarrow p_1 \wedge p_2 \wedge \ldots \wedge p_n)$. In a CNF form this is equivalent to a set of $\bigwedge_i (w_{p_i}, a \rightarrow p_i)$. In our representation, we have weight w_p for each axiom. Effect axioms can be defined similarly. $E \in Eff$ is $(w_e, a \rightarrow e_1 \wedge e_2 \wedge \ldots \wedge e_n)$. Precondition axioms dictate the facts at the time when the actions are conducted, while the effect axioms state the facts on the next time slot.

STRIPS planning domains can be straightforwardly translated into our representation. For any given STRIPS operator definition $O = \{\text{PRE}, \text{ADD}, \text{DEL}\}$, we can create precondition axiom $(1, O \rightarrow \text{PRE})$, one effect axiom $(1, O \rightarrow \text{ADD} \wedge \neg\text{DEL})$. For example, for the "pickup" action of Blocksworld domain, STRIPS defines it as "pickup(x)" = $\{\{\text{(armempty)}, \text{(ontable x)}, \text{(clear x)}\}, \{\text{(holding x)}\}, \{\text{(armempty)}, \text{(ontable x)}, \text{(clear x)}\}\}$ In our scheme, we have two axioms for "pickup".

- (1, "pickup(x)"$\rightarrow$ (armempty) $\wedge$ (ontable x) $\wedge$ (clear x)), for precondition axiom
- (1, "pickup(x)"$\rightarrow$ (holding x) $\wedge$ $\neg$(armempty) $\neg$(ontable x) $\neg$(clear x)) , for effect axiom

When the weights of all the axioms are one, it means that we have a complete and deterministic planning domain. When the weights are not one, it means that we have incomplete or probabilistic domain. So, in this representation, incomplete domain knowledge and probabilistic domain are indistinguishable. Indeed, if one does not have complete domain knowledge, a natural planning strategy is finding a plan that is most likely to achieve the goal with the current domain knowledge, although the target domain is deterministic domain.

Incorporating Background Knowledge

When complete domain knowledge is given, there is no need of background knowledge for domain invariant. For example, "at most one block can be on top of some other block" is an invariant of Blocksworld domain. This invariant is maintained by the carefully written set of action definitions. Instead, it could be easier to provide a planner with explicit domain invariant than write a set of action definitions.

We consider explicit domain knowledge in our representation. In the definition of $\mathbb{D}$, $\mathbb{B}$ is there for this purpose. $\mathbb{B}$ can specify domain invariant or property of solution plans. Standard planners that assume complete domain knowledge don't have to use any domain invariant, since they are automatically maintained through legal actions. However, for model-lite planners, background knowledge that defines domain invariant can be very useful. For example, when there is an action which breaks the domain invariant, then the planner does not consider the action during the plan composition. In the following planning section, our proposed planning approach can directly account for $\mathbb{B}$.

Control Knowledge $\mathbb{B}$ can also specify the properties of good plans. There are such obvious properties for many planning domains. E.g., for Blocksworld, "don't take put-down action after pickup action" is such property. A human can easily provide this property and planner should exploit it. This is somewhat similar to TL-Plan's approach (Bacchus & Kabanza 2000). The difference is that we can update the background knowledge especially when the input control knowledge is uncertain. Again, our proposed planning approach can naturally consider background control knowledge.

Planning

We define a planning problem for a domain $\mathbb{D}$ with a tuple of objects and facts (O, I, G). O is a set of objects for the problem, thus the set of facts and the set of actions are defined by grounding $\mathcal{P}$ and $\mathcal{A}$ with O. I is the initial state and G is a set of goal facts that must be achieved at the end of the solution plan. For deterministic domain, the solution is any sequence of actions that achieves the goal though we prefer shorter sequences. For probabilistic or incomplete domains, the criteria is different. We like to have a sequence of actions that achieves the goal with maximum likelihood or a tree-like plan that suggests actions based on the observation, or contingent plan. In our case we seek the former plan.

Planning as MPE

The objective of planning in this work is finding a plan that achieves a given goal with maximum probability under the given domain definition $\mathbb{D}$.

$$\max_{a_1,\ldots,a_T} \Pr(a_1 \ldots a_T | I^1, G^T) \quad (1)$$

The superscripts of I and G in the above equation means the time point. Equation 1 assumes that the planning horizon is T. Thus, the planning problem here is to find a sequence of actions that maximizes the probability of the actions given the initial state and the goal facts. This is very similar to MPE (most probable explanation) solution to the problem of finding sequence of T actions given the evidence of initial state and goal facts. Indeed, (Verma & Rao 2006) has studied MPE aspect of MDP (markov decision processes) planning problems. However the work has remained in a purely enumerated non-factored state space scenario. In our study, we will extend the aspect in relational graphplan framework, thus exploiting the relational structural property. We induce MPE aspect through novel view of plangraph as bayesian network.

Plangraph as Bayesian Network

To find a most probable plan for the given problem, we start with constructing a probabilistic plangraph (Blum & Furst 1995). Plangraph is a leveled graph, where fact level and action level are interleaved. The 1st level has initial facts I and actions available with I. Then the next level of facts are constructed with the addition of the added facts of the actions. Facts that were true at i-th level are transferred to the next level through NO-OP action. Deleted facts are considered as Mutex relations between facts and actions. Levels

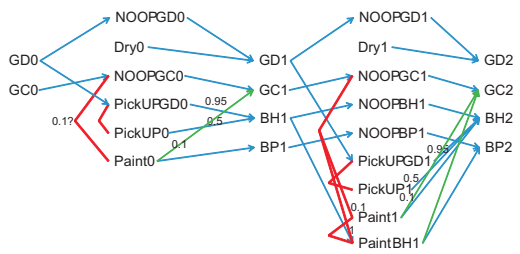

Figure 1: A Plangraph for Gripper Problem: a 2 level plangraph is drawn. First, third and fifth columns correspond to facts level 1, 2 and 3. Second and Fourth columns correspond to actions levels 1 and 2. The edges between facts and actions level are drawn from precondition and effect axiom. The numbers indicate the probability of the axioms. Edges with no number indicates probability 1. Edges within action levels indicate mutex relation

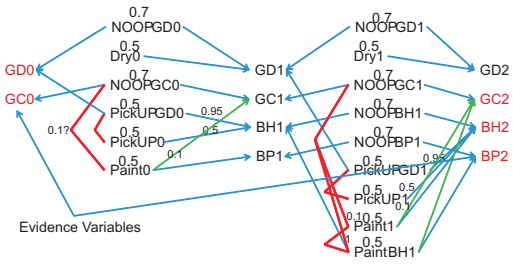

Figure 2: A Bayesian Network for Gripper Problem: A translation of a plangraph to a bayesian network. Arrows of the edges are now toward facts level, reflecting the conditional probability of facts given the actions in the axioms.

in the plangraph are increased until the saturation of facts in the fact level or dynamic mutex happens. For deterministic domains, solution plan is found by searching for truth assignment of action levels that achieves the goal and does not violate the mutex constraints.

In our case, we draw plangraph out of precondition axiom and effect axiom, starting with the initial state. Then we label each edge with the probabilities associated with the axiom. Figure 1 shows an example plangraph for a Gripper domain. This domain has 5 actions, pickup-gripperdry, pickup, dry, paint-blockinhand and paint. The pickup action achieves block-in-hand (BH) with probability 0.5 and precondition is not gripper-dry (GD). Pickup-gripperdry (Pickup-GD) achieves block-in-hand (BH) with probability 0.95 and precondition is gripper-dry (GD). Dry action achieves gripper-dry (GD). Paint-blockinhand (Paint-BH) action achieves block-painted (BP) with probability 1 and achieves not gripper-clean (GC) with probability 1. Precondition is block-in-hand (BH). Paint action achieves block-painted (BP) with probability 1 and achieves not gripper-clean (GC) with probability 0.1. Precondition is not block-in-hand (BH). [1]

The problem of this gripper domain is achieving block-in-hand (BH), and block-painted (BP), starting from gripper-dry (GD) and gripper-clean (GC).

Previous works on probabilistic plangraph sought ways to propagate probabilities through the plangraph (Blum & Langford 1999). Instead we take the view of (Verma & Rao 2006) and attempt to find the most plausible truth assignment of action variables. To do this, we first interpret the plangraph as a bayesian network. The probabilities of facts in fact level are described with conditional probability with adjacent action level actions. The actions in the action levels are given a prior probability distribution of 0.5. Figure 2 shows a bayesian network version of the plangraph.

Below we summarize the construction of conditional probability table for the plangraph.

- Actions in Action Level: We give probability 0.5 for actions in the action levels. Alternatively, we can give some prior probability by solving determinized version of the problem as FF-Replan did (Yoon, Fern, & Givan 2007). In this case, we give higher probability than 0.5 for the actions that are in the solutions of the determinized version of the problem.
- Precondition Axiom: We give the probability of facts of the precondition of the actions as specified in the precondition axiom. E.g., for a precondition axiom (0.95 (pickup x) $\rightarrow$ (clear x)), we give 0.95 for (clear a)^t when (pickup a)^t is true. If not, we give 0.5.
- No-op actions: No-op actions are treated in differently. No-op actions do not have precondition axioms. Their truth values are decided by the truth value of the corresponding facts of the current level. For example (noop-(clear a))^t is true with probability 1, when (clear a)^t is true. The superscript here specifies the level of the fact or action in the plangraph.
- Effect Axiom: We give the probability of facts of the effects of the actions as specified in the effect axiom. E.g., for an effect axiom (0.95 (pickup x) $\rightarrow$ (holding x)), we give 0.95 for (holding a)$^{t+1}$ when (pickup a)^t is true. No-op actions are treated in the same way. The effect for the no-op actions have probability 1. For example (clear a)$^{t+1}$ is true with probability 1, when (noop-(clear a)^t is true.
- Static Mutex: Two actions are mutex at the same level if

[1] Actions with conditional effects are separated into different actions for each condition from original probabilistic Gripper domian

they delete added facts or precondition of the other. Two actions are mutex between consecutive level if one action in the earlier level deletes the precondition of the other. In our case, the mutex is probabilistic. For two actions A and B, and when deleted facts of A deletes added facts of B, we compute the probability of the mutex as follows, $p = \max_{d \in}$Deleted Facts in A$_{,a \in}$Added Facts in B $P_d \times P_a$. Thus the probability of the mutex is the maximum of the occurrence of the mutex. Other style of static mutexes are similarly defined and asserted as conditional probability.

- Dynamic Mutex: We propagate probability through levels of the plangraph only for dynamic Mutex. Two facts are mutex when all the actions that achieve them are mutex with each other. The probability of the facts being mutex is then the maximum of the mutex probabilities of the involved actions.
- Background Knowledge: Background knowledge is asserted in the facts level as well as actions level. For example, if the background knowledge for a Blocksworld states that (holding x) $\rightarrow \neg$ (armempty), we generate the conditional probability with probability 1. Thus (armempty) is false with probability one when (holding a) is true is generated. Typically this involves generating multiple probability table entries (proportional to the number of objects) for one background knowledge.

  TL-Plan like background knowledge can also be asserted as they are available. For example, a Blocksworld control rule (pickup x) $\rightarrow \neg$ (putdown x), ("don't putdown a block right after picking it up from the table"), can be asserted as, (putdown a)$^{t+1}$ is false with probability 1, when (pickup a)^t is true. Here interesting part is that background knowledge can be asserted with probability. This is particularly useful when the background knowledge itself is learned or given with some uncertainty.
- Initial State and Goal Facts: Initial State and Goal Facts are asserted with probability one as they are the evidence variables. $\forall f \in I, f^1$ True and $\forall g \in G, g^T$ True, when the horizon is T.

Probabilistic planning as MAXSAT

We have made a bayesian network out of plangraph with previous subsection's probability assignment. The remaining task is finding an MPE solution for the bayesian network. Here the evidence variables are initial state and goal facts in the plangraph. The solution is the most probable truth assignment to the rest of the propositions (actions and facts) in the plangraph, given the probabilities. The exact MPE solution technique would not scale well enough. (Park 2002) has shown how one can translate a bayesian network to a weighted MAXSAT problem. We take the approach and we generate a weighted MAXSAT problem. Then we can use any available approximate weighted MAXSAT solvers. The summary of planning procedure is in Figure 3.

Current Experimental Test Status We have tested our idea on small instance (the Gripper problem) with hand-coding of the bayesian network of the problem and used Maxwalksat solver. It has shown promising results. We are currently implementing the system and investigating potential impact of approximate solutions to the WCNF.

Planning $(\mathbb{D}, I, G, O)$
// Incomplete Domain Definition $\mathbb{D}$, Initial State I, Goal G, Objects O
1. $Plg \leftarrow$ Build-a-probabilistic-plangraph $(\mathbb{D}, I, G, O)$
   Plg is a probabilistic plangraph for the given problem
2. $BN \leftarrow$ Build-probability-table (Plg)
   BN is a bayesian network for the given problem
3. $WCNF \leftarrow$ Convert-to-Weighted-CNF (BN)
   $WCNF$ is a final translation of the given problem to a weighted CNF
4. $V \leftarrow$ solve-using-weighted-MAXSAT $(WCNF)$
   S is truth assignment of the variables
5. $S \leftarrow$ translate-the-solution (V)
   finally translate the truth assignment of the variables into a plan, ignoring truth assignment of fact levels and focusing on action levels
6. return S

Figure 3: Incomplete Domain Planning using weighted MAXSAT

Learning

When incomplete domain model is given, planning can be achieved with the planning algorithm that we provided. However, as experience on the domain is gathered, we can update the input domain model, and do a better planning. We update the domain model with state-action pair trajectories. The source of the trajectories can be either from pure wandering, during plan execution or human demonstration.

Given state-action pair trajectories, we consider two aspects of learning. First, we just update the weights of the precondition and effect axioms. We can use readily available package for this purpose, e.g., Alchemy (Richardson & Domingos 2006). Second, we construct new precondition and effect axioms, considering background knowledge.

Given a state-action trajectory $\mathbb{J} = \{(s_1, a_1), \ldots, (s_n, a_n)\}$, we construct training data for each axiom set as follows.

1. Precondition Axiom Training Data : $\{(s_1, a_1), \ldots, (s_n, a_n)\}$. Each state and action pair is a valid world example for precondition axioms.
2. Effect Axiom Training Data : $\{(a_i, s_{i+1})\}$. Each action and the next state pair is a valid world example of effect axioms.
3. Control Knowledge (if the trajectory is from a solution plan) : $\{(a_i, a_{i+1})\}$, or $\{(s_1, a_1), \ldots, (s_n, a_n)\}$. The former one is for learning temporal constraints on action selection, and the latter is for learning reactive control.

Weight Update

Consider an effect axiom $a =$ (w, (pickup x) $\rightarrow$ (holding x)) of Blocksworld. We construct training examples as in the

second item in the above enumeration. For example, $e = \{$ (pickup a), (ontable b), (holding a) $\ldots\}$ is one instance of training data, where the first element is the action and the rest is the observed facts of the next state. When e is presented, the axiom a can be applied and w is updated with $w = w + s$, where s is learning step size (positive example). When another instance, $e' = \{$ (pickup a), (ontable b), $\ldots\}$, is presented, w is reduced by s, $w = w - s$ (negative example). The same weight update principle is applied to precondition axiom and control knowledge axioms learning.

There are readily available machine learning package for this purpose, especially Alchemy (Richardson & Domingos 2006). Such package produces weights for the input axioms, given a set of training data. Thus, we can directly use such package, with our training data construction. The idea of separation of precondition and effect axioms enables this straightforward machine learning solution.

When a comprehensive set of axioms or control knowledge is presented, we can simply use weight update method as mentioned. Wrong axioms or unnecessary axioms would get close to 0 weight after enough training data and updates, and necessary axioms would get close to the probability of the precondition or effect axioms. However, it is hard to have a comprehensive set of axioms. Next, we consider how we parsimoniously construct candidate axioms with background knowledge.

Constructing Candidate Axioms

Since we can update weights of the axioms with trajectories, we might just naively enumerate all the possible candidate axioms. Let's just consider precondition axiom first. Effect axiom or control knowledge can be treated similarly. For every action a, we can enumerate all the possible axioms of the form $a(x, y) \rightarrow p(x, y)$, with all the available predicates and variable matching.

This will create unnecessary axioms and make the learning slower and especially will make planning much slower, since it will create the plangraph where every potential facts and actions appear at the first few levels, thus there is no advantage of considering plangraph. We need to generate axioms more parsimoniously and we can do this with background knowledge.

In Blocksworld, consider we are given a background knowledge that states that "a block cannot be both on the table and on some other block", (ontable x) $\leftrightarrow \neg$ (on x y). When we observe a "(pickup a)" action and a state fact "(ontable a)", we could immediately cut out *(pickup x)* $\rightarrow$ *(on x y)* from the candidate axiom set and keep *(pickup x)* $\rightarrow$ *(ontable x)*. Thus, with background knowledge we can eliminate axioms that conflict with knowledge and keep the axiom set small. Again, this is particularly important for later planning.

Learning Test Status We have conducted domain learning on Blocksworld with Alchemy software. With 100 state-action pairs, Alchemy found higher weights on correct axioms, and lower weights on incorrect axioms.

Discussion

Planning operator learning has been studied by several researchers. Two of the recent and closest approaches are (Yang, Wu, & Jiang 2007) and (Zettlemoyer, Pasula, & Kaelbling 2005). (Yang, Wu, & Jiang 2007) learns preconditions and effects of domain operators only from initial state, goal state and plan. Compared to the approach, our learning approach can leverage any partial domain knowledge and produce probabilistic domain model, which is well suited for incomplete domain planning. (Zettlemoyer, Pasula, & Kaelbling 2005) learns very complex operator format, using *deictic* representation. This study did not consider using available partial domain knowledge.

Machine learning approach to Model-lite planning is nothing new at all. Indeed, model-free reinforcement learning is the simplest and classic approach to applying machine learning techniques to model-lite planning. The weakness of reinforcement learning in general is its scalability. Prodigy or Soar (Russell & Norvig 1995) systems have developed techniques that can update the relational operators as they interact with environments. These systems focused on learning preconditions of the operators. Well specified preconditions can act as a strategy for the target domain. These studies kept on learning deterministic operators or preconditions and did not find probabilistic effects or preconditions.

Our planning algorithm is closely related to many probabilistic planning techniques. There are planning techniques that can handle different form of uncertainties, e.g., conformant planning and probabilistic planning techniques. Conformant planning can output robust plan when observing is not allowed. Probabilistic planning techniques are targeted to handling uncertain outcomes but typically they don't consider uncertain preconditions. Our framework in this paper can uniformly consider uncertainties of preconditions and effects together. Maxplan work (Majercik & Littman 1998) has developed a variant of SAT encoding technique from a probabilistic planning problem. Maxplan used probabilistic variables to represent probabilistic effects, while we used weight on clauses to represent probabilistic effects. Pgraphplan (Blum & Langford 1999) has used plangraph structure directly and propagated the probabilities through the levels of the plangraph. Instead we convert the probabilistic plangraph to a weighted MAXSAT problem. Another difference is the treatment of Mutexes. Pgraphplan treated any pair of actions as mutex if any of the probabilistic effects interferes each other. In our case, the mutex is weighted proportional to the probabilities of the occurrence.

One interesting future approach is interleaving the learning and planning part. Thus, the system is given a problem and the system builds model as it is attempting to solve the problem. (Chang & Amir 2006) has approached this scenario through logical filtering idea, but the planning approach was limited to deterministic domain and did not exploit the probabilistic aspect. (Kearns & Singh 1998) showed theoretical property of the approach but did not give any realistic implementation on relational representation domain.

Our approach in this work is also capable of learning strategy. As briefly mentioned in previous section, con-

trol knowledge can be represented as a form of background knowledge that constrains the action choices. Learning and using such knowledge in SAT has been studied by (Huang, Selman, & Kautz 2000). In that work, the learned knowledge has been used as deterministic rules and can hurt the planning performance, as the learning is not perfect. In our framework, imperfect learning is reflected on the weight of the rules and can be seamlessly used as weighted clauses.

References

Bacchus, F., and Kabanza, F. 2000. Using temporal logics to express search control knowledge for planning. *Journal of Artificial Intelligence* 16:123–191.

Bartk, R., and McCluskey, L. 2005. International knowledge engineering competition.

Blum, A., and Furst, M. 1995. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, 1636–1642.

Blum, A., and Langford, J. 1999. Probabilistic planning in the graphplan framework. In *ECP*, 319–332.

Chang, A., and Amir, E. 2006. Goal achievement in partially known, partially observable domains. In *ICAPS*.

DARPA. 2006. Integrated learning program.

De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*.

Huang, Y.-C.; Selman, B.; and Kautz, H. 2000. Learning declarative control rules for constraint-based planning. In *Proceedings of the 17th International Conference on Machine Learning*, 415–422. Morgan Kaufmann, San Francisco, CA.

Kambhampati, S. 2007. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain theories. In *AAAI*.

Kearns, M., and Singh, S. 1998. Near-optimal reinforcement learning in polynomial time. In *Proc. 15th International Conf. on Machine Learning*, 260–268. Morgan Kaufmann, San Francisco, CA.

Majercik, S. M., and Littman, M. L. 1998. MAXPLAN: A new approach to probabilistic planning. In *Artificial Intelligence Planning Systems*, 86–93.

Park, J. D. 2002. Using weighted MAX-SAT engines to solve MPE. In *Eighteenth national conference on Artificial intelligence*, 682–687. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Richardson, M., and Domingos, P. 2006. Markov logic network. *Machine Learning*.

Russell, S., and Norvig, P. 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Verma, D., and Rao, R. 2006. Goal-based imitation as probabilistic inference over graphical models. In *Proc. 18th NIPS*.

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

Yoon, S.; Fern, A.; and Givan, R. 2007. FF-replan: A baseline for probabilistic planning. In *ICAPS*.

Zettlemoyer, L.; Pasula, H.; and Kaelbling, L. 2005. Learning planning rules in noisy stochastic worlds. In *AAAI*.