Action-Model Acquisition from Noisy Plan Traces

Hankz Hankui Zhuo[a] **and Subbarao Kambhampati**[b]
[a]Dept. of Computer Science, Sun Yat-sen University, Guangzhou, China
zhuohank@mail.sysu.edu.cn
[b]Dept. of Computer Science and Engineering, Arizona State University, US
rao@asu.edu

Abstract

There is increasing awareness in the planning community that the burden of specifying complete domain models is too high, which impedes the applicability of planning technology in many real-world domains. Although there have been many learning approaches that help automatically creating domain models, they all assume plan traces (training data) are *correct*. In this paper, we aim to remove this assumption, allowing plan traces to be with noise. Compared to collecting large amount of correct plan traces, it is much easier to collect noisy plan traces, e.g., we can directly exploit sensors to help collect noisy plan traces. We consider a novel solution for this challenge that can learn action models from noisy plan traces. We create a set of random variables to capture the possible correct plan traces behind the observed noisy ones, and build a graphical model to describe the physics of the domain. We then learn the parameters of the graphical model and acquire the domain model based on the learnt parameters. In the experiment, we empirically show that our approach is effective in several planning domains.

1 Introduction

Most work in planning assumes that complete domain models (a domain model is composed of a set of action models) are given as input in order to synthesize plans. However, there is increasing awareness that building domain models presents steep challenges for domain creators. Indeed, recent work in web-service composition [Bertoli *et al.*, 2010; Hoffmann *et al.*, 2007] and work-flow management [Blythe *et al.*, 2004] suggests that dependence on complete domain models can well be the real bottle-neck inhibiting applications of current planning technology.

Attempts have been made to design systems to automatically learn domain models from pre-specified (or pre-collected) plan traces (a plan trace is composed of an action sequence with zero or more partial states). For example, Amir [Amir, 2005] presented a tractable and exact technique for learning action models known as Simultaneous Learning and Filtering (SLAF). Yang et al. [Yang *et al.*, 2007; Zhuo *et al.*, 2010] proposed to learn STRIPS action models [Fikes and Nilsson, 1971] from plan traces with partially observed states. These systems, however, are all based on the assumption that actions in plan traces are observed correctly.

In many real-world applications, however, actions in plan traces are often observed incorrectly. For instance, consider an urban disaster rescue environment, where there are heterogeneous agents such as fire fighters, commanders, polices, victims, etc., conducting search and rescue actions; these actions are observed and collected via various sensors. Noise[1] is inevitably introduced into plan traces when some sensors are occasionally damaged. Another scenario is there have been many activity/plan recognition approaches [Kautz and Allen, 1986; Bui, 2003; Ramirez and Geffner, 2010] that can be exploited to output plan traces. These output plan traces are often rife with noise. It is therefore important to develop an approach capable of tolerating noise in plan traces.

We assume that each action has a possibility to be noisy in plan traces. Based on this assumption, we propose a graphical model based policy-gradient approach to learn action models. We call this approach `AMAN`, which stands for **A**ction-**M**odel **A**cquisition from **N**oisy plan traces. Specifically, `AMAN` 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, we first exploit the observed noisy plan traces to predict correct plan traces and the domain model based on the graphical model, and then execute the correct plan traces to calculate the reward of the predicted correct plan traces according to a predefined reward function (which will be defined in Section 4.3). We update the predicted plan traces and domain model based on the reward. We iteratively perform the above-mentioned steps until a given number of iterations is reached. As a result, we output the predicted domain model.

As we mentioned above, there have been systems that can learn action models from correctly observed plan traces, such as `ARMS` [Yang *et al.*, 2007]. However, extending these previous systems to handle plan traces with noisy actions is nontrivial. For example, extending `ARMS` to tolerate noisy actions

[1]Note that an action is considered to be *noisy* if it is mistakenly observed, e.g., action "pickup" is mistakenly observed as "putdown", which suggests "putdown" is a noisy action.

in plan traces requires adding or removing constraints from ARMS to tolerate noise. However, either way of adding or removing constraints is hard to capture the information about the probabilistic distribution of noisy actions in plan traces in the MAX-SAT [LI *et al.*, 2007] framework which is used by ARMS. Considering the difficulty in extending previous systems, we explore a novel approach which is based on graphical model, as addressed above.

We organize the paper as follows. We first review related work, and then present the formal definition of our learning problem. After that, we give the detailed description of the AMAN algorithm. Finally, we evaluate AMAN in three planning domains, and conclude the paper with future work.

2 Related Work

Learning action models from plan traces has had a long history. Previous research efforts differ mainly on whether plan traces consist of actions as well as intermediate states (c.f. [Gil, 1994]) or only actions (c.f. [Yang *et al.*, 2007; Zhuo *et al.*, 2010; 2011]). While the later assumption makes the problem harder than the former, in both cases, whatever is observed is assumed to be observed perfectly. In contrast, our work focuses on learning from noisy (imperfectly observed) plan traces. While there has been previous work on learning probabilistic action models (e.g. [Pasula *et al.*, 2007] and [Zettlemoyer *et al.*, 2005]), they too assume non-noisy plan observations. Work on plan recognition (e.g. [Zhuo *et al.*, 2012; Bui, 2003]) has of course considered recognizing plans from noisy/missing observations, but they typically assume that they know the action model. Our problem is however significantly complicated by the fact that we need to *learn* the action model given only noisy plan traces. One recent effort that does take imperfectly sensed plan traces into consideration is that by Mourao et al. [Mourao *et al.*, 2012], who allow for noisy observations of the intermediate states. They however assume that the actions are observed without noise.

3 Preliminaries and Problem Formulation

A planning problem is defined by $\langle s_0, g, m\rangle$, where s_0 is an initial state, g is a goal, and m is called a domain model that is composed of a set of action models. An initial state or a goal is composed of a set of propositions. An action model is defined by $\langle a, Pre, Add, Del\rangle$, where a is an action name with zero or more variables (as parameters), Pre, Add and Del are *precondition* list, *add* list and *delete* list, respectively. In the paper we will use $Pre(a)$ to denote a list of preconditions of action a, likewise for $Add(a)$ and $Del(a)$. Note that we follow the STRIPS language definition [Fikes and Nilsson, 1971] action models. We call an action with zero or more parameters (variables) an *action schema*, a set of which is denoted by $\bar{A}$. We denote the set of action instances of a planning problem by A. A solution to a planning problem is a plan, which is composed of an action sequence $\langle a_1, a_2, \ldots, a_n\rangle$, where $a_i \in A$.

Let $\mathcal{T}^o = \{\langle s_0, a_1^o, \ldots, a_n^o, g\rangle\}$ be a set of observed plan traces, where $a_i^o \in A^o$ is an observed action, and A^o is a set of possible observed actions. We assume the empty action $noop$ (that does nothing) is also an element of A^o. For simplicity, we assume $A^o = \{noop\} \cup A$. An observed action sequence is allowed to be with noise, i.e., some actions in a plan trace may be mistakenly observed. Our problem can be defined by: given as input a set of observed plan traces $\mathcal{T}^o$, our approach outputs a domain model m that best explains the observed plan traces. We assume that there is a correct plan trace t for each observed plan trace t^o, where the length of t (the number of actions in t) is the same as t^o, i.e., $|t| = |t^o|$. An example of our learning problem can be found in Table 1. The gray parts of the input plan traces are mistakenly observed actions (i.e., noisy actions), e.g., "(pickup A)" is a noisy action that should be "(unstack A C)".

Table 1: An example of our learning problem.

Input:	plan trace #1: s_0^1, *(pickup A)*, (putdown A), *noop*, (stack B A), g^1 plan trace #2: s_0^2, (pickup a), *(putdown a)*, (pickup b), (stack b a), g^2 (— *other plan traces omitted* —)
Output:	(:**action** (pickup ?x - block) (:precondition (ontable ?x) (handempty) (clear ?x)) (:effects (and (holding ?x) (not (ontable ?x)) (not (clear ?x)) (not (handempty))))) (— *other action models omitted* —)

s_0^1:{(on A C) (ontable C) (clear A) (ontable B) (clear B) (handempty)}; g^1:{(ontable A) (on B A)}.
s_0^2: {(ontable a) (ontable b) (clear a) (clear b) (ontable c) (clear c) (handempty)}; g^2: {(on a c) (on b a)}.

4 The AMAN Approach

Consider a domain model m to be a set of action models. Given a set of action schemas and a set of predicates, we can generate a finite set of domain models (denoted by $M = \{m\}$) by enumerating all possible combinations of action schemas and predicates. Our objective is to find a domain model $m^* \in M$ that best explains the observed plan traces. To do this, we perform the following steps. We first extract action schemas $\bar{A}$ and predicates P from the input plan traces, and build a set of domain models M with $\bar{A}$ and P. We then build a graphical model according to the causal relations between states s_i, correct actions a_i, observed actions a_i^o (probably noisy) and domain models m. We finally learn the graphical model using the input plan traces and output the target domain model m^*. In summary, we first build a set of candidate domain models M. We then build a graphical model to capture the domain physics. After that we learn the graphical model to predict the best domain model m^*. In the following subsections, we describe the idea in detail.

4.1 Building Candidate Domain Models

In this subsection, we extract action schemas and predicates from input plan traces. We first scan each action in plan traces and substitute its instantiated parameters with their corresponding variables to build an action schema. In this way we can build a set of action schemas $\bar{A}$. Likewise, we scan each initial state and goal in plan traces to collect a set of predicates P. *As an example, from the two plan traces in Table 1, we can build action schemas $\bar{A}$={(pick ?x - block), (put-down ?x - block), noop, (stack ?x - block ?y - block)}, and predicates P={(on ?x - block ?y - block) (ontable ?x - block) (clear ?x - block) (handempty)}.*

With action schemas $\bar{A}$ and predicates P, we further build all possible candidate domain models M based on STRIPS description [Fikes and Nilsson, 1971]. Each domain model $m \in M$ should satisfy the following constraints (which would help reduce the model space):

- For each action a and predicate p in the model m, where p's parameters are included by a's parameters, if p is an adding effect of a, then p should not be a deleting effect at the same time, i.e., $p \in Add(a) \to p \notin Del(a)$.
- For each action a and predicate p, where p's parameters are included by a's parameters, if p is a precondition of a, then p should not be an adding effect at the same time, i.e., $p \in Pre(a) \to p \notin Add(a)$.
- For each action a, there is at least one precondition and one effect (adding or deleting effect), i.e., $Pre(a) \neq \emptyset \wedge (Add(a) \cup Del(a)) \neq \emptyset$.

In this way, we can enumerate all the possible domain models M and view them as a set of candidate domain models.

4.2 Modeling Domain Dynamics

In this subsection we build a graphical model to capture the relations between state s_i, action a_{i+1} (the correct action), action a^o_{i+1} (the observed action) and domain model m. Intuitively, the probability of action a_{i+1} depends on its previous state and the domain model since a_{i+1} depends on whether it can be applied on its previous state s_i according to the domain model m; and it also depends on its corresponding observed action since the more similar to the observed action a^o_{i+1} the candidate correct action a_{i+1} is, the more likely the correct action a_{i+1} happens. Furthermore, the probability of new state s_{i+1} depends on its previous state s_i, its previous action a_{i+1} and the current domain model m, since s_{i+1} is attained by applying a_{i+1} to s_i and changing s_i to s_{i+1} according to m (or the effect of a_{i+1}). We thus build a graphical model as shown in Figure 1. Note that in Figure 1 s_n is the same as g. Note that in Figure 1, s_0, a^o_i and s_n can be attained

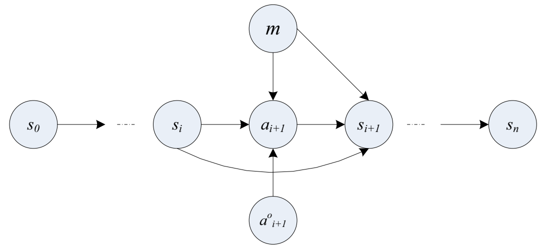

Figure 1: The graphical model of our learning problem.

from the input observed plan trace$t^o = \langle s_0, a^o_1, \ldots, a^o_n, s_n\rangle$, while s_i $(0 < i < n)$, a_i and m are "hidden" variables whose values need to be "guessed".

To predict the correct plan traces and action models, we define the conditional probability of $t = \langle s_0, a_1, \ldots, a_n, s_n\rangle$ and action model m given $t^o = \langle s_0, a^o_1, \ldots, a^o_n, s_n\rangle$ as follows:

$$p(t, m|t^o) \propto p(t, m, t^o). \tag{1}$$

Based on the graphical model in Figure 1, $p(t, m, t^o)$ can be calculated by

$$p(t, m, t^o) = p(m)p(s_0) \prod_{i=0}^{n-1} \Big\{ p(a_{i+1}|s_i, a^o_{i+1}, m) \times p(s_{i+1}|a_{i+1}, s_i, m) \Big\}. \tag{2}$$

Since we focus on deterministic models, there is only one state s_{i+1} achieved when applying action a_{i+1} to state s_i based domain model m. As a result, $p(s_{i+1}|a_{i+1}, s_i, m)$ is 1, and 0 for other states s'_{i+1}. Thus, Equation (2) can be reduced to:

$$p(t, m, t^o) = p(m)p(s_0) \prod_{i=0}^{n-1} p(a_{i+1}|s_i, a^o_{i+1}, m). \tag{3}$$

We assume that each domain model $m \in M$ is associated with a weight w_m (the weights of all domain models are denoted by $\vec{w} = \langle w_m\rangle_{m\in M}$), and $p(m)$ is defined by

$$p(m) = \frac{w_m}{\sum_{m'\in M} w_{m'}}.$$

According to Equations (1) and (3), we have

$$p(t, m|t^o) \propto \frac{w_m}{\sum_{m'\in M} w_{m'}} p(s_0) \prod_{i=0}^{n-1} p(a_{i+1}|s_i, a^o_{i+1}, m). \tag{4}$$

We model the distribution $p(a_{i+1}|s_i, a^o_{i+1}, m)$ over the action space in a log-linear fashion [Pietra *et al.*, 1997; Lafferty *et al.*, 2001], giving us the flexibility to incorporate a diverse range of features. With this representation, the distribution is:

$$p(a_{i+1}|s_i, a^o_{i+1}, m) = \frac{\exp(\vec{\theta} \cdot \mathbf{f}(a_{i+1}, s_i, a^o_{i+1}, m))}{\sum_{a'_{i+1}} \exp(\vec{\theta} \cdot \mathbf{f}(a'_{i+1}, s_i, a^o_{i+1}, m))}, \tag{5}$$

where $\mathbf{f}(a_{i+1}, s_i, a^o_{i+1}, m) = \langle f_1, \ldots, f_k\rangle$ is an k-dimensional feature representation, and $\vec{\theta}$ is a k-vector that includes parameters to be determined. k is the number of features that are used to model the domain. We use 10 features in our experiment to describe domains. The following are example features we exploit (note that it is possible to exploit more features for further extension):

- f_1 *(applicability of a_{i+1}): the value of this feature is one if a_{i+1} can be applied on s_i according to m; otherwise, it is zero;*
- f_2 *(applicability of a^o_{i+1}): the value of this feature is one if a^o_{i+1} can be applied on s_i according to m; otherwise, it is zero;*

- f_3 *(number of parameters): the value of this feature is one if* a_{i+1} *and* a^o_{i+1} *have the same number of parameters; otherwise, it is zero;*
- f_4 *(shared objects): the value of this feature is one if* a_{i+1} *and* a^o_{i+1} *share some common objects; otherwise it is zero;*
- *Features from* f_5 *to* f_{10} *are extracted to describe the similarity between* a_{i+1} *and* a^o_{i+1} *regarding their precondition lists,* add *lists,* delete *lists, as well as their semantics in real applications, such as "directions" of these actions.*

With features defined, the distribution $p(t, m|t^o)$ only depends on parameters $\vec{\theta}$ and $\vec{w}$. In the following, we will present the details of learning parameters $\vec{\theta}$ and $\vec{w}$.

4.3 Learning Parameters

With the correct plan trace t predicted, we aim to maximize the expected value function based on the predefined reward function:

$$V_{\vec{\theta},\vec{w}}(t^o) = E_{p(t,m|t^o)}[r(t, m)], \tag{6}$$

where $r(t, m)$ is a reward function that specifies the reward attained after executing t, m. Note that $r(t, m)$ can be defined in various ways. In our experiment, we define $r(t, m)$ by

$$r(t, m) = \lambda_a(t, m) \cdot \lambda_p(t, m), \tag{7}$$

where $\lambda_a(t, m)$ is the percentage of actions successfully executed in t and $\lambda_p(t, m)$ is the percentage of propositions in the goal of t achieved after the last successfully executed action. The intuition behind this definition is that the more percentage of actions are successfully executed, the more likely the predicted plan trace t is correct; and the more percentage of propositions in goal are achieved, the more likely the plan trace t is correct.

The learning problem is to find the parameters $\vec{\theta}$ and $\vec{w}$ that maximize $V_{\vec{\theta},\vec{w}}(t^o)$ from Equation 6. Although there is no closed form solution, policy gradient algorithms [Sutton *et al.*, 2000] are capable of estimating the parameters $\vec{\theta}$ and $\vec{w}$ by performing stochastic gradient ascent. The gradient of $V_{\vec{\theta},\vec{w}}$ is approximated by interacting with the environment, and the resulting reward is used to update the estimate of $\vec{\theta}$ and $\vec{w}$. Policy gradient algorithms optimize a non-convex objective and are only guaranteed to find a local optimum. However, as we will see, they scale to large state spaces and can perform well in practice.

To find the parameters $\vec{\theta}$ and $\vec{w}$ that maximize the objective, we compute the derivative of $V_{\vec{\theta},\vec{w}}$. Expanding the derivative according to the product rule, we have (note that $p(s_0)$ is a prior probability which is viewed as 1):

$$\frac{\partial}{\partial\vec{\theta}} V_{\vec{\theta},\vec{w}}(t^o) = E_{p(t,m|t^o)}\left[r(t, m)\frac{\partial}{\partial\vec{\theta}} \log p(t, m|t^o)\right], \tag{8}$$

and

$$\frac{\partial}{\partial\vec{w}} V_{\vec{\theta},\vec{w}}(t^o) = E_{p(t,m|t^o)}\left[r(t, m)\frac{\partial}{\partial\vec{w}} \log p(t, m|t^o)\right]. \tag{9}$$

Expanding the inner partial derivative from Equations (8) and (9) we observe that

$$\frac{\partial}{\partial\vec{\theta}} \log p(t, m|t^o) = \sum_i \frac{\partial}{\partial\vec{\theta}} \log p(a_{i+1}|s_i, a^o_{i+1}, m) = \sum_i \{f(a_{i+1}, s_i, a^o_{i+1}, m) - \sum_{a'_{i+1}} [f(a'_{i+1}, s_i, a^o_{i+1}, m) \, p(a'_{i+1}|s_i, a^o_{i+1}, m)]\}, \tag{10}$$

and

$$\frac{\partial}{\partial\vec{w}} \log p(t, m|t^o) = \frac{\partial}{\partial\vec{w}} \log \frac{w_m}{\sum_{m' \in M} w_{m'}}. \tag{11}$$

We denote $\delta_1 = \frac{\partial}{\partial\vec{\theta}} \log p(t, m|t^o)$ and $\delta_2 = \frac{\partial}{\partial\vec{w}} \log p(t, m|t^o)$. Then we have

$$\vec{\theta} = \vec{\theta} + r(t, m) \cdot \delta_1, \tag{12}$$

and

$$\vec{w} = \vec{w} + r(t, m) \cdot \delta_2. \tag{13}$$

We can see that Equations (10) and (11) are straightforward to compute. However, the complete derivatives of $V_{\vec{\theta},\vec{w}}$ in Equations (8) and (9) are intractable to compute since the expectation requires enumerating all possible plan traces t (as well as domain models m). Fortunately, policy gradient algorithms employ stochastic gradient ascent by computing an estimate of the expectation using just a subset of possible plan traces. Specifically, we perform the sampling procedure by the following two steps:

1. We draw a sample domain model m from M based $p(m)$ provided that $\vec{w}$ was beforehand computed or initialized.
2. We then draw a sample plan trace t by (a). first drawing action a_{i+1} based on the distribution $p(a_{i+1}|s_i, a^o_{i+1}, m)$ provided that $\vec{\theta}$ and s_i were computed or initialized and m was drawn in the first step, (b). and then computing a new state s_{i+1} by directly applying a_{i+1} to s_i based on m.

4.4 Overview of AMAN

As an overview of AMAN, we show the learning procedure in Algorithm 1.

Our AMAN algorithm framework belongs to a family of policy gradient algorithms, which have been successfully applied to complex problems (e.g., robot control [Ng *et al.*, 2003], natural language processing [Branavan *et al.*, 2012]). Our formulation is unique in how it learns action models in the planning community.

5 Experiments

5.1 Training Data and Criterion

We evaluate our AMAN algorithm in three planning domains *blocks*[2], *depots*[3] and *driverlog*[2] which are IPC (International Planning Competition) benchmark domains. We generate 200 noisy plan traces (training data) by the following steps:

[2]http://www.cs.toronto.edu/aips2000/aips-2000datafiles.tgz
[3]http://planning.cis.strath.ac.uk/competition/

Algorithm 1 An overview of our AMAN framework.

Input: a set of plan traces: $T^o = \{t^o\}$.
Output: a domain model m^*.

```
1: initialize $\vec{\theta}$ and $\vec{w}$ with random values, and set an iteration number $R$;
2: for $r = 1$ to $R$ do
3:     for each $t^o = \langle s_0, a_1^o, \ldots, a_n^o, g\rangle \in T$ do
4:         sample $t$ and $m$ based on $p(t, m|t^o)$;
5:         calculate the reward $r(t, m)$ based on Equation (7);
6:         update $\vec{\theta}$ and $\vec{w}$ based on Equations (12) and (13);
7:     end for
8: end for
9: $m^* = \arg\max_{m \in M} w_m$;
10: return $m^*$;
```

- We first generate a set of *correct* plan traces $\{t\}$ by using a planner (such as FF[4]) to solve randomly generated planning problems, assuming correct action models (ground truth action models) are available.
- We then randomly assign values to $\vec{\theta}$, and calculate the conditional distribution $p(a_{i+1}^o|s_i, a_{i+1}, m)$ of observed actions given state s_i, action a_{i+1}, $\vec{\theta}$ and ground truth action models m (similar to Equation (5)). Note that s_i can be generated by executing actions in the first step and a_{i+1} is a correct action generated by the first step.
- We finally sample 200 noisy plan traces $\{t^o\}$ based on $p(a_{i+1}^o|s_i, a_{i+1}, m)$. Each noisy plan trace t^o corresponds to a correct plan trace t generated by the first step.

We define the accuracy of our AMAN algorithm by comparing its learned action models with the artificial action models which are viewed as the ground truth. We define the error rate of the learning result by calculating the missing and extra predicates of the learned action models. Specifically, for each learned action model a, if a precondition of a does not exist in the ground-truth action model, then the number of errors increases by one; if a precondition of the ground-truth action model does not exist in a's precondition list, then the number of errors also increases by one. As a result, we have the total number of errors of preconditions with respect to a. We define the error rate of preconditions (denoted as $Err_{pre}(a)$) as the proportion of the total number of errors among all the possible preconditions of a, that is,

$$Err_{pre}(a) = \frac{\text{the total number of errors of preconditions}}{\text{all the possible precondition of } a}.$$

Likewise, we can calculate the error rates of adding effects and deleting effects of a, and denote them as $Err_{add}(a)$ and $Err_{del}(a)$ respectively. Furthermore, we define the error rate of all the action models m (denoted as $Err(m)$) as the average of $Err_{pre}(a)$, $Err_{add}(a)$ and $Err_{del}(a)$ for all the actions a in m, that is,

$$Err(m) = \frac{1}{|m|} \sum_{a \in m} \frac{Err_{pre}(a) + Err_{add}(a) + Err_{del}(a)}{3},$$

[4]http://fai.cs.uni-saarland.de/hoffmann/ff.html

and define the accuracy as $Acc = 1 - Err(m)$. Note that domain model m is composed of a set of action models.

5.2 Experimental Results

We evaluated our AMAN algorithm in the following aspects. We first studied how the accuracy of the model learned by AMAN varies with respect to increasing number of iterations R. In doing this, we considered both noisy inputs and noiseless inputs. We are interested in seeing whether AMAN's accuracy on noisy inputs converges to its accuracy on noiseless traces. We also compared the performance of AMAN on noiseless input traces to ARMS [Yang *et al.*, 2007] one of the state-of-the-art action model learning systems (which assumes noiseless input traces). Finally, we studied the running time of AMAN. In all experiments we ran AMAN three times to calculate an average of accuracies.

Accuracy on noisy input traces w.r.t. #iterations R

To test the change of accuracies with respect to the number of iterations R, we varied R from 500 to 2500 and calculated their corresponding accuracies. We tested two cases "*without noise*" and "*with noise*" in plan traces, where "*without noise*" suggests all plan traces are all correct and we do not need to sample t (as done below in the experiment comparing to ARMS), and "*with noise*" suggests we directly run Algorithm 1 with any change. The results are shown in Figure 2.

From Figure 2, we can see that the accuracies of both cases generally become larger when the value of R becomes larger. This indicates that high-quality domain models (with respect to maximally satisfying plan traces) are enhanced, which are guaranteed by the convergence property of AMAN. We also observe that the accuracies of the case "*with noise*" become more and more close to "*without noise*" as the value of R becomes larger. This is because the distribution of noise in plan traces becomes more and more close to the real distribution when the number of iterations increasing. In other words, the value of $\vec{\theta}$ is close to the real value that we randomly fixed for generating noise of plan traces in Section 5.1. By observing the results of the three testing domains, we can see the accuracies of the case with noise are larger than 0.8 when R is set to be 1500.

Comparison between AMAN and ARMS on noiseless inputs

Since the "*without noise*" case is also solved by ARMS, we would like to compare the "*without noise*" case of AMAN to ARMS to see the performance of AMAN. Since ARMS does not allow noisy actions in input plan traces, we fed both AMAN and ARMS with plan traces *without noise* to make the comparison be fair. In this way, we do not need to sample t in step 4 of Algorithm 1 since $t = t^o$ and do not need to update $\vec{\theta}$ in step 6 either. We set the number of iterations R in AMAN to be 1500. We varied the number of plan traces from 40 to 200 to see the changes of accuracies. The results are shown in Figure 3.

From Figure 3, we can see that in all three domains, the accuracies of both AMAN and ARMS are very close to each other indicating that neither dominates the other. The rationale is AMAN exploits the reward function $r(t, m)$ to enhance the domain model that can best explain the plan traces, whose aim is

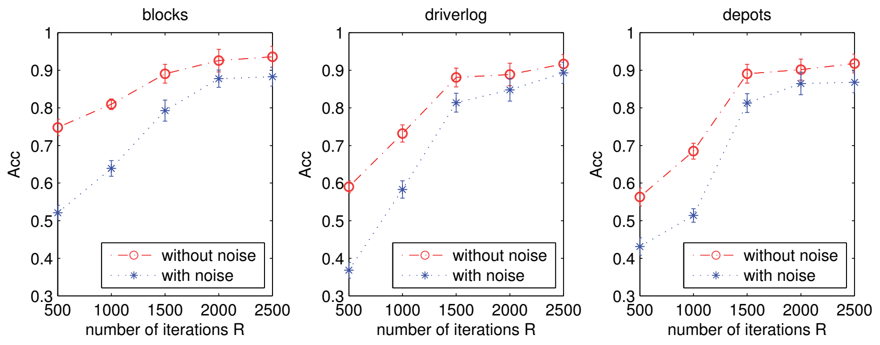

Figure 2: The accuracies with respect to different number of iterations.

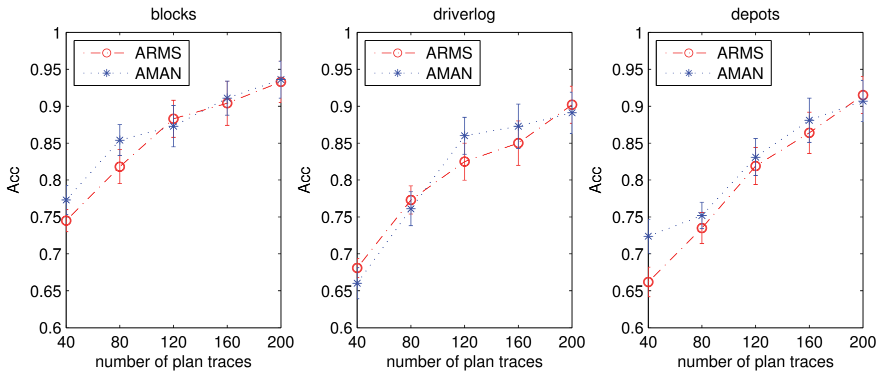

Figure 3: The comparison between AMAN and ARMS.

the same as ARMS that exploits a set of constraints to acquire a domain model that best satisfies the input plan traces. From Figure 3, we also observe that the accuracies generally become higher when the number of plan traces increasing. This is consistent with our intuition since the more plan traces we have access to, the more information can be exploited to improve the learning result.

Running time

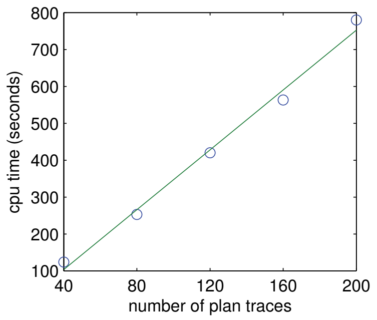

Figure 4: The running time w.r.t. number of plan traces.

To show the running time of AMAN, we ran AMAN in the *blocks* domain with respect to different number of plan traces as input, setting R to be 1500. The result is shown in Figure 4. As can be seen from the figure, the running time generally increases linearly with respect to the number of input plan traces. We fitted the relation between the number of plan traces and the running time to verify this fact. For example, the fitting result in Figure 5 is $y = 4.0550x - 58.6000$.

6 Conclusion

In this paper we propose a novel approach to learning action models for planning. Different from previous learning systems, our approach allows noisy actions in the input plan traces, which is the first approach in the planning community that is capable of handling noisy actions. Currently we assume that the noisy plan traces have been collected by previous systems such as plan recognition systems. In the future, we would like to extend AMAN to learning action models from real-world data, such as text documents [Branavan *et al.*, 2012], from which actions with a possibility to be noisy can be learnt.

Acknowledgements: Hankz Hankui Zhuo thanks Natural Science Foundation of Guangdong Province of China (No. S2011040001869), Research Fund for the Doctoral Program of Higher Education of China (No. 20110171120054), and Guangzhou Science and Technology Project (No. 2011J4300039) for the support of this research. Kambhampati's research is supported in part by the ARO grant W911NF-13-1-0023, the ONR grants N00014-13-1-0176, N00014-09-1-0017 and N00014-07-1-1049, and the NSF grant IIS201330813.

References

[Amir, 2005] E. Amir. Learning partially observable deterministic action models. In *Proceedings of IJCAI-05*, pages 1433–1439, 2005.

[Bertoli *et al.*, 2010] Piergiorgio Bertoli, Marco Pistore, and Paolo Traverso. Automated composition of web services via planning in asynchronous domains. *Artificial Intelligence Journal*, 174(3-4):316–361, 2010.

[Blythe *et al.*, 2004] J. Blythe, E. Deelman, and Y. Gil. Automatically composedworkflows for grid environments. *IEEE Intelligent Systems*, 19(4):16–23, 2004.

[Branavan *et al.*, 2012] S.R.K. Branavan, Nate Kushman, Tao Lei, and Regina Barzilay. Learning high-level planning from text. In *Proceedings of ACL-12*, 2012.

[Bui, 2003] Hung H. Bui. A general model for online probabilistic plan recognition. In *Proceedings of IJCAI-03*, 2003.

[Fikes and Nilsson, 1971] R. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence Journal*, pages 189–208, 1971.

[Gil, 1994] Yolanda Gil. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Proceedings of ICML-94*, pages 87–95, 1994.

[Hoffmann *et al.*, 2007] Jrg Hoffmann, Piergiorgio Bertoli, and Marco Pistore. Web service composition as planning, revisited: In between background theoriesandinitial state uncertainty. In *Proceedings of AAAI-07*, 2007.

[Kautz and Allen, 1986] Henry A. Kautz and James F. Allen. Generalized plan recognition. In *Proceedings of AAAI-86*, 1986.

[Lafferty *et al.*, 2001] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML-01*, pages 282–289, 2001.

[LI *et al.*, 2007] Chu Min LI, Felip Manya, and Jordi Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321–359, October 2007.

[Mourao *et al.*, 2012] Kira Mourao, Luke Zettlemoyer, Ronald P. A. Petrick, and Mark Steedman. Learning strips operators from noisy and incomplete observations. In *Proceedings of UAI-12*, pages 614–623, 2012.

[Ng *et al.*, 2003] Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. Autonomous helicopter flight via reinforcement learning. In *Proceedings of NIPS-03*, 2003.

[Pasula *et al.*, 2007] Hanna M. Pasula, Luke S. Zettlemoyer, and Leslie Pack Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research*, 29:309–352, 2007.

[Pietra *et al.*, 1997] Stephen Della Pietra, Vincent J. Della Pietra, and John D. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–39, 1997.

[Ramirez and Geffner, 2010] Miquel Ramirez and Hector Geffner. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of AAAI-10*, 2010.

[Sutton *et al.*, 2000] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of NIPS-00*, page 10571063, 2000.

[Yang *et al.*, 2007] Qiang Yang, Kangheng Wu, and Yunfei Jiang. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171:107–143, February 2007.

[Zettlemoyer *et al.*, 2005] Luke S. Zettlemoyer, Hanna M. Pasula, and Leslie Pack Kaelbling. Learning planning rules in noisy stochastic worlds. In *Proceedings of AAAI-05*, 2005.

[Zhuo *et al.*, 2010] Hankz Hankui Zhuo, Qiang Yang, Derek Hao Hu, and Lei Li. Learning complex action models with quantifiers and implications. *Artificial Intelligence*, 174(18):1540 – 1569, 2010.

[Zhuo *et al.*, 2011] Hankz Hankui Zhuo, Qiang Yang, Rong Pan, and Lei Li. Cross-domain action-model acquisition for planning via web search. In *Proceedings of ICAPS-11*, pages 298–305, 2011.

[Zhuo *et al.*, 2012] H.H. Zhuo, Q. Yang, and S. Kambhampati. Action-model based multi-agent plan recognition. In *Advances in Neural Information Processing Systems 25*, pages 377–385, 2012.