Nominated for AAMAS 2016
Best Student Paper Award

Discovering Underlying Plans Based on Distributed Representations of Actions

Xin Tian
Dept of Computer Science
Sun Yat-Sen University
Guangzhou, China. 510006
tianxin1860@gmail.com

Hankz Hankui Zhuo*
Dept. of Computer Science
Sun Yat-Sen University
Guangzhou, China. 510006
zhuohank@mail.sysu.edu.cn

Subbarao Kambhampati
Dept. of Computer Sci. & Eng.
Arizona State University
Tempe, Arizona, US
rao@asu.edu

ABSTRACT

Plan recognition aims to discover target plans (i.e., sequences of actions) behind observed actions, with history plan libraries or domain models in hand. Previous approaches either discover plans by maximally "matching" observed actions to plan libraries, assuming target plans are from plan libraries, or infer plans by executing domain models to best explain the observed actions, assuming complete domain models are available. In real world applications, however, target plans are often not from plan libraries and complete domain models are often not available, since building complete sets of plans and complete domain models are often difficult or expensive. In this paper we view plan libraries as corpora and learn vector representations of actions using the corpora; we then discover target plans based on the vector representations. Our approach is capable of discovering underlying plans that are not from plan libraries, without requiring domain models provided. We empirically demonstrate the effectiveness of our approach by comparing its performance to traditional plan recognition approaches in three planning domains.

1. INTRODUCTION

As computer-aided cooperative work scenarios become increasingly popular, human-in-the-loop planning and decision support has become a critical planning challenge (c.f. [5, 6, 17]). An important aspect of such a support [13] is recognizing what plans the human in the loop is making, and provide appropriate suggestions about their next actions [1]. Although there is a lot of work on plan recognition, much of it has traditionally depended on the availability of a complete domain model [21, 31]. As has been argued elsewhere [13], such models are hard to get in human-in-the-loop planning scenarios. Here, the decision support systems have to make themselves useful without insisting on complete action models of the domain. The situation here is akin to that faced by search engines and other tools for computer supported cooperate work, and is thus a significant departure for the "planning as pure inference" mindset of the automated planning community. As such, the problem calls for plan recognition with "shallow" models of the domain (c.f. [11]), that can be easily learned automatically.

There has been very little work on learning such shallow models to support human-in-the-loop planning. Some examples include the work on Woogle system [6] that aimed to provide support to humans in web-service composition. That work however relied on very primitive understanding of the actions (web services in their case) that consisted merely of learning the input/output types of individual services.

In this paper, we focus on learning more informative models that can help recognize the plans under construction by the humans, and provide active support by suggesting relevant actions. To drive this process, we need to learn shallow models of the domain. We propose to adapt the recent successes of word-vector models [18] in language to our problem. Specifically, we assume that we have access to a corpus of previous plans that the human user has made. Viewing these plans as made up of action words, we learn word vector models for these actions. These models provide us a way to induce the distribution over the identity of each unobserved action. Given the distributions over individual unobserved actions, we use an expectation-maximization approach to infer the joint distribution over all unobserved actions. This distribution then forms the basis for action suggestions.

We will present the details of our approach, and will also empirically demonstrate that it does capture a surprising amount of structure in the observed plan sequences, leading to effective plan recognition. We further compare its performance to traditional plan recognition techniques, including one that uses the same plan traces to learn the STRIPS-style action models, and use the learned model to support plan recognition.

2. PROBLEM DEFINITION

A plan library, denoted by $\mathcal{L}$, is composed of a set of plans $\{p\}$, where p is a sequence of actions, i.e., $p = \langle a_1, a_2, \ldots, a_n \rangle$ where a_i, $1 \leq i \leq n$, is an action name (without any parameter) represented by a string. For example, a string *unstack-A-B* is an action meaning that *a robot unstacks block A from block B*. We denote the set of all possible actions by $\bar{\mathcal{A}}$ which is assumed to be known beforehand. For ease of presentation, we assume that there is an empty action, ϕ, indicating an unknown or not observed action, i.e., $\mathcal{A} = \bar{\mathcal{A}} \cup \{\phi\}$. An observation of an *unknown* plan $\tilde{p}$ is denoted by $\mathcal{O} = \langle o_1, o_2, \ldots, o_M \rangle$, where $o_i \in \mathcal{A}$, $1 \leq i \leq M$, is either an action in $\bar{\mathcal{A}}$ or an empty action ϕ indicating the corresponding action is missing or not observed. Note that $\tilde{p}$ is not necessarily in the plan library $\mathcal{L}$, which makes the plan recognition problem more challenging, since matching the observation to the plan library will not work any more.

We assume that the human is making a plan of at most length M. We also assume that at any given point, the planner is able to observe $M - k$ of these actions. The k unobserved actions might either be in the suffiix (i.e., yet to be formed part) of the plan, or in the middle (due to observational gaps). Our aim is to suggest, for

*Corresponding author.

Appears in: *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, S. Marsella (eds.), May 9–13, 2016, Singapore.*

each of the k unobserved actions, m possible choices–from which the user can select the action. (Note that we would like to keep m small, ideally close to 1, so as not to overwhelm the user). Accordingly, we will evaluate the effectiveness of the decision support in terms of whether or not the user's best/intended action is within the suggested m actions.

Specifically, our recognition problem can be represented by a triple $\Re = (\mathcal{L}, \mathcal{O}, \mathcal{A})$. The solution to $\Re$ is to discover the unknown plan $\tilde{p}$ that best explains $\mathcal{O}$ given $\mathcal{L}$ and $\mathcal{A}$. An example of our plan recognition problem in the *blocks*[1] domain is shown below.

Example: A plan library $\mathcal{L}$ in the *blocks* domain is assumed to have four plans as shown below:

plan 1: *pick-up-B stack-B-A pick-up-D stack-D-C*
plan 2: *unstack-B-A put-down-B unstack-D-C put-down-D*
plan 3: *pick-up-B stack-B-A pick-up-C stack-C-B pick-up-D stack-D-C*
plan 4: *unstack-D-C put-down-D unstack-C-B put-down-C unstack-B-A put-down-B*

An observation $\mathcal{O}$ of action sequence is shown below:

observation: *pick-up-B ϕ unstack-D-C put-down-D ϕ stack-C-B ϕ ϕ*

Given the above input, our DUP algorithm outputs plans as follows:

pick-up-B stack-B-A unstack-D-C put-down-D pick-up-C stack-C-B pick-up-D stack-D-C

Although the "plan completion" problem seems to differ superficially from the traditional "plan recognition" problem, we point out that many earlier works on plan recognition do in fact evaluate their recognition algorithms in terms of completion tasks, e.g., [22, 28, 31]. While these earlier efforts use different problem settings, taking either a plan library or action models as input, they share one common characteristic: they all aim to look for a plan that can best explain (or complete) the observed actions. This is exactly the same as our problem we aim to solve.

3. OUR DUP ALGORITHM

Our DUP approach to the recognition problem $\Re$ functions by two phases. We first learn vector representations of actions using the plan library $\mathcal{L}$. We then iteratively sample actions for unobserved actions o_i by maximizing the probability of the unknown plan $\tilde{p}$ via the EM framework. We present DUP in detail in the following subsections.

3.1 Learning Vector Representations of Actions

Since actions are denoted by a name string, actions can be viewed as words, and a plan can be viewed as a sentence. Furthermore, the plan library $\mathcal{L}$ can be seen as a corpus, and the set of all possible actions $\mathcal{A}$ is the vocabulary. We thus can learn the vector representations for actions using the Skip-gram model with hierarchical softmax, which has been shown an efficient method for learning high-quality vector representations of words from unstructured corpora [18].

The objective of the Skip-gram model is to learn vector representations for predicting the surrounding words in a sentence or document. Given a corpus $\mathcal{C}$, composed of a sequence of training words $\langle w_1, w_2, \ldots, w_T \rangle$, where $T = |\mathcal{C}|$, the Skip-gram model maximizes the average log probability

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} \log p(w_{t+j}|w_t) \quad (1)$$

[1]http://www.cs.toronto.edu/aips2000/

where c is the size of the training window or context.

The basic probability $p(w_{t+j}|w_t)$ is defined by the hierarchical softmax, which uses a binary tree representation of the output layer with the K words as its leaves and for each node, explicitly represents the relative probabilities of its child nodes [18]. For each leaf node, there is an unique path from the root to the node, and this path is used to estimate the probability of the word represented by the leaf node. There are no explicit output vector representations for words. Instead, each inner node has an output vector $v'_{n(w,j)}$, and the probability of a word being the output word is defined by

$$p(w_{t+j}|w_t) = \prod_{i=1}^{L(w_{t+j})-1} \Big\{\sigma(\mathbb{I}(n(w_{t+j}, i+1) = child(n(w_{t+j}, i))) \cdot v_{n(w_{t+j},i)} \cdot v_{w_t})\Big\}, \quad (2)$$

where

$$\sigma(x) = 1/(1+\exp(-x)).$$

$L(w)$ is the length from the root to the word w in the binary tree, e.g., $L(w) = 4$ if there are four nodes from the root to w. $n(w,i)$ is the ith node from the root to w, e.g., $n(w,1) = root$ and $n(w, L(w)) = w$. $child(n)$ is a fixed child (e.g., left child) of node n. v_n is the vector representation of the inner node n. v_{w_t} is the input vector representation of word w_t. The identity function $\mathbb{I}(x)$ is 1 if x is true; otherwise it is -1.

We can thus build vector representations of actions by maximizing Equation (1) with corpora or plan libraries $\mathcal{L}$ as input. We will exploit the vector representations to discover the unknown plan $\tilde{p}$ in the next subsection.

3.2 Maximizing Probability of Unknown Plans

With the vector representations learnt in the last subsection, a straightforward way to discover the unknown plan $\tilde{p}$ is to explore all possible actions in $\bar{\mathcal{A}}$ such that $\tilde{p}$ has the highest probability, which can be defined similar to Equation (1), i.e.,

$$\mathcal{F}(\tilde{p}) = \sum_{k=1}^{M}\sum_{-c\leq j\leq c, j\neq 0} \log p(w_{k+j}|w_k) \quad (3)$$

where w_k denotes the kth action of $\tilde{p}$ and M is the length of $\tilde{p}$. As we can see, this approach is exponentially hard with respect to the size of $\bar{\mathcal{A}}$ and number of unobserved actions. We thus design an approximate approach in the Expectation-Maximization framework to estimate an unknown plan $\tilde{p}$ that best explains the observation $\mathcal{O}$.

To do this, we introduce new parameters to capture "weights" of values for each unobserved action. Specifically speaking, assuming there are X unobserved actions in $\mathcal{O}$, i.e., the number of ϕs in $\mathcal{O}$ is X, we denote these unobserved actions by $\bar{a}_1, ..., \bar{a}_x, ..., \bar{a}_X$, where the indices indicate the order they appear in $\mathcal{O}$. Note that each $\bar{a}_x$ can be any action in $\bar{\mathcal{A}}$. We associate each possible value of $\bar{a}_x$ with a weight, denoted by $\bar{\Gamma}_{\bar{a}_x,x}$. $\bar{\Gamma}$ is a $|\bar{\mathcal{A}}| \times X$ matrix, satisfying

$$\sum_{o\in\bar{\mathcal{A}}} \bar{\Gamma}_{o,x} = 1,$$

where $\bar{\Gamma}_{o,x} \geq 0$ for each x. For the ease of specification, we extend $\bar{\Gamma}$ to a bigger matrix with a size of $|\bar{\mathcal{A}}| \times M$, denoted by Γ, such that $\Gamma_{o,y} = \bar{\Gamma}_{o,x}$ if y is the index of the xth unobserved action in $\mathcal{O}$, for all $o \in \bar{\mathcal{A}}$; otherwise, $\Gamma_{o,y} = 1$ and $\Gamma_{o',y} = 0$ for all $o' \in \bar{\mathcal{A}} \wedge o' \neq o$. Our intuition is to estimate the unknown plan $\tilde{p}$ by selecting actions with the highest weights. We thus introduce

the weights to Equation (2), as shown below,

$$p(w_{k+j}|w_k) = \prod_{i=1}^{L(w_{k+j})-1} \Big\{\sigma(\mathbb{I}(n(w_{k+j}, i+1) = child(n(w_{k+j}, i))) \cdot av_{n(w_{k+j},i)} \cdot bv_{w_k}\Big\}, \quad (4)$$

where $a = \Gamma_{w_{k+j},k+j}$ and $b = \Gamma_{w_k,k}$. We can see that the impact of w_{k+j} and w_k is penalized by weights a and b if they are unobserved actions, and stays unchanged, otherwise (since both a and b equal to 1 if they are observed actions). We redefine the objective function as shown below,

$$\mathcal{F}(\tilde{p}, \Gamma) = \sum_{k=1}^{M} \sum_{-c \le j \le c, j \neq 0} \log p(w_{k+j}|w_k), \quad (5)$$

where $p(w_{k+j}|w_k)$ is defined by Equation (4). The only parameters needed to be updated are Γ, which can be easily done by gradient descent, as shown below,

$$\Gamma_{o,x} = \Gamma_{o,x} + \delta \frac{\partial \mathcal{F}}{\partial \Gamma_{o,x}}, \quad (6)$$

if x is the index of unobserved action in $\mathcal{O}$; otherwise, $\Gamma_{o,x}$ stays unchanged, i.e., $\Gamma_{o,x} = 1$. Note that δ is a learning constant.

With Equation (6), we can design an EM algorithm by repeatedly sampling an unknown plan according to Γ and updating Γ based on Equation (6) until reaching convergence (e.g., a constant number of repetitions is reached).

An overview of our DUP algorithm is shown in Algorithm 1. In Step 2 of Algorithm 1, we initialize $\Gamma_{o,k} = 1/M$ for all $o \in \bar{\mathcal{A}}$, if k is an index of unobserved actions in $\mathcal{O}$; and otherwise, $\Gamma_{o,k} = 1$ and $\Gamma_{o',k} = 0$ for all $o' \in \bar{\mathcal{A}} \wedge o' \neq o$. In Step 4, we view $\Gamma_{\cdot,k}$ as a probability distribution, and sample an action from $\bar{\mathcal{A}}$ based on $\Gamma_{\cdot,k}$ if k is an unobserved action index in $\mathcal{O}$. In Step 5, we only update $\Gamma_{\cdot,k}$ where k is an unobserved action index. In Step 6, we linearly project all elements of the updated Γ to between 0 and 1, such that we can do sampling directly based on Γ in Step 4. In Step 8, we simply select $\bar{a}_x$ based on

$$\bar{a}_x = \arg\max_{o \in \bar{\mathcal{A}}} \Gamma_{o,x},$$

for all unobserved action index x.

Algorithm 1 Framework of our DUP algorithm

Input: plan library $\mathcal{L}$, observed actions $\mathcal{O}$
Output: plan $\tilde{p}$

1: learn vector representation of actions
2: initialize $\Gamma_{o,k}$ with $1/M$ for all $o \in \bar{\mathcal{A}}$, when k is an unobserved action index
3: **while** the maximal number of repetitions is not reached **do**
4: sample unobserved actions in $\mathcal{O}$ based on Γ
5: update Γ based on Equation (6)
6: project Γ to [0,1]
7: **end while**
8: select actions for unobserved actions with the largest weights in Γ
9: **return** $\tilde{p}$

Our DUP algorithm framework belongs to a family of policy gradient algorithms, which have been successfully applied to complex problems, e.g., robot control [19], natural language processing [3]. Our formulation is unique in how it recognizes plans, in comparison to the existing methods in the planning community.

Note that our current study shows that even direct application of word vector learning methods provide competitive performance for plan completion tasks. We believe we can further improve the performance by using the planning specific structural information in the EM phase. In other words, if we are provided with additional planning structural information as input, we can exploit the structural information to filter candidate plans to be recognized in the EM procedure.

4. EXPERIMENTS

In this section, we evaluate our DUP algorithms in three planning domains from International Planning Competition, i.e., blocks[1], depots[2], and driverlog[2]. To generate training and testing data, we randomly created 5000 planning problems for each domain, and solved these planning problems with a planning solver, such as FF[3], to produce 5000 plans. We then randomly divided the plans into ten folds, with 500 plans in each fold. We ran our DUP algorithm ten times to calculate an average of accuracies, each time with one fold for testing and the rest for training. In the testing data, we randomly removed actions from each testing plan (i.e., $\mathcal{O}$) with a specific percentage ξ of the plan length. Features of datasets are shown in Table 1, where the second column is the number of plans generated, the third column is the total number of words (or actions) of all plans, and the last column is the size of vocabulary used in all plans.

Table 1: Features of datasets

domain	#plan	#word	#vocabulary
blocks	5000	292250	1250
depots	5000	209711	2273
driverlog	5000	179621	1441

We define the accuracy of our DUP algorithm as follows. For each unobserved action $\bar{a}_x$ DUP suggests a set of possible actions S_x which have the highest value of $\Gamma_{\bar{a}_x,x}$ for all $\bar{a}_x \in \bar{\mathcal{A}}$. If S_x covers the *truth* action a_{truth}, i.e., $a_{truth} \in S_x$, we increase the number of correct suggestions g by 1. We thus define the accuracy acc as shown below:

$$acc = \frac{1}{T} \sum_{i=1}^{T} \frac{\#\langle correct\text{-}suggestions \rangle_i}{K_i},$$

where T is the size of testing set, $\#\langle correct\text{-}suggestions \rangle_i$ is the number of correct suggestions for the ith testing plan, K_i is the number of unobserved actions in the ith testing plan. We can see that the accuracy acc may be influenced by S_x. We will test different size of S_x in the experiment.

State-of-the-art plan recognition approaches with plan libraries as input aim at finding a plan from plan libraries to best explain the observed actions [9], which we denote by MatchPlan. We develop a MatchPlan system based on the idea of [9] and compare our DUP algorithm to MatchPlan with respect to different percentage of unobserved actions ξ and different size of suggestion set S_x. Another baseline is action-models based plan recognition approach [22] (denoted by PRP, short for Plan Recognition as Planning). Since we do not have action models as input in our DUP algorithm, we exploited the action model learning system ARMS [26] to learn action models from the plan library and feed the action models to the PRP approach. We call this hybrid plan recognition approach ARMS+PRP. To learn action models, ARMS requires

[2]http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/long03a-html/JAIRIPC.html

[3]https://fai.cs.uni-saarland.de/hoffmann/ff.html

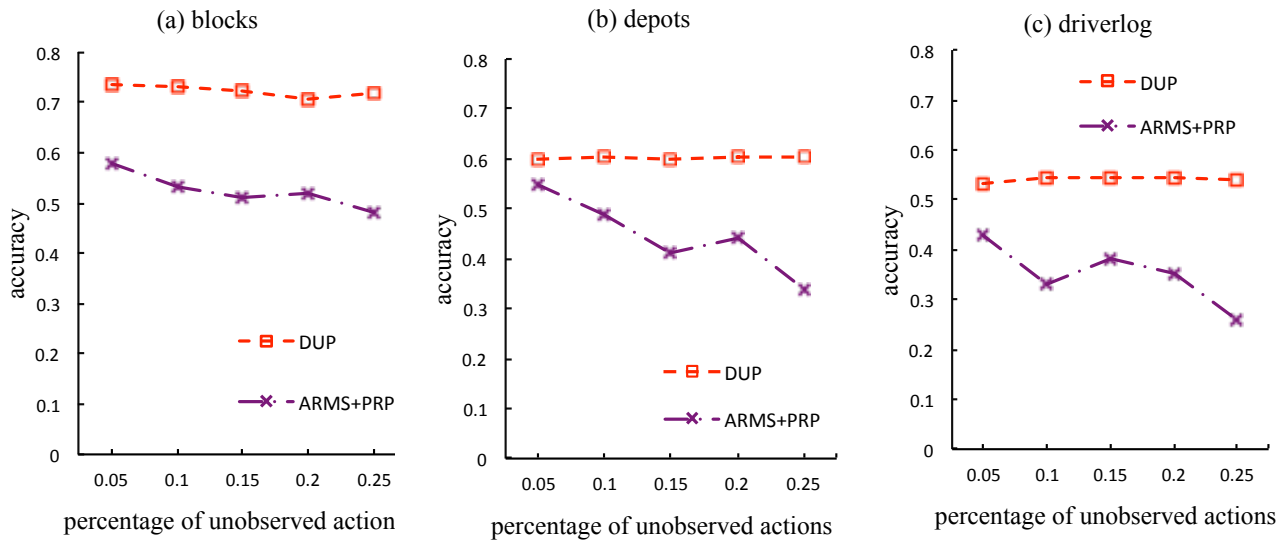

Figure 1: Accuracies of DUP and ARMS+PRP with respect to different percentage of unobserved actions

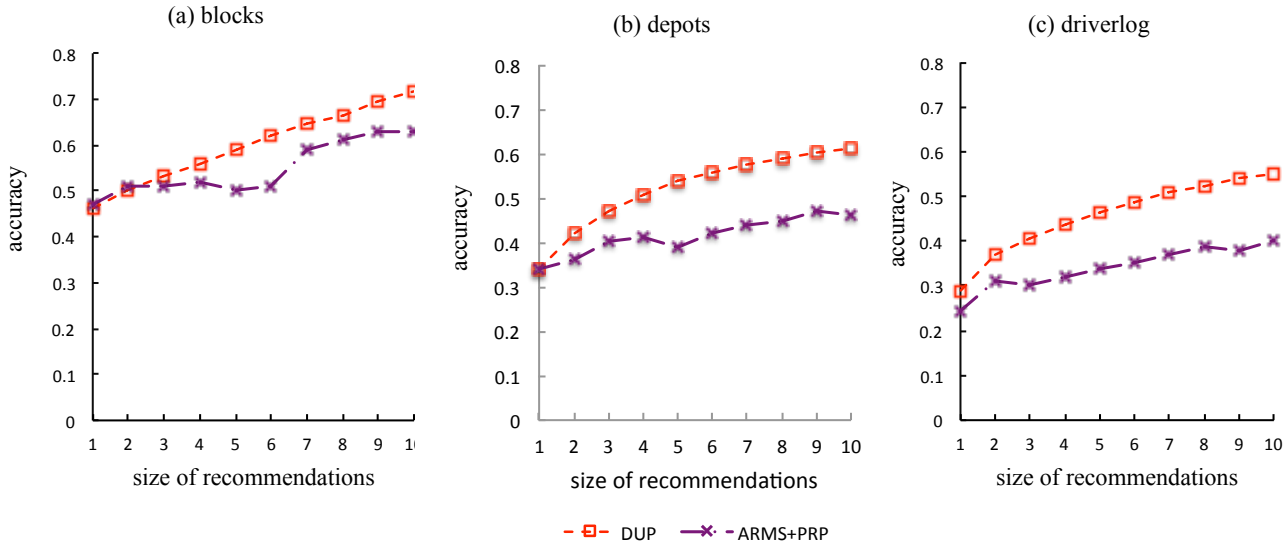

Figure 2: Accuracies of DUP and ARMS+PRP with respect to different size of recommendations

state information of plans as input. We thus added extra information, i.e., initial state and goal of each plan in the plan library, to ARMS+PRP. In addition, PRP requires as input a set of candidate goals $\mathcal{G}$ for each plan to be recognized in the testing set, which was also generated and fed to PRP when testing. In summary, the hybrid plan recognition approach ARMS+PRP has more input information, i.e., initial states and goals in plan library and candidate goals $\mathcal{G}$ for each testing example, than our DUP approach.

4.1 Comparison between DUP and ARMS+PRP

We first compare our DUP algorithm to ARMS+PRP to see the advantage of DUP. We varied the percentage of unobserved actions and the size of recommended actions to see the change of accuracies, respectively. The results are shown below.

4.1.1 *Varying Percentage of Unobserved Actions*

In this experiment we would like to see the change of accuracies of both our DUP algorithm and ARMS+PRP with respect to ξ in $\mathcal{O}$. We set the window of training context c in Equation (1) to be three, the number of iterations in Algorithm 1 to be 1500, the size of recommendations to be ten, and the learning constant δ in Equation (6) to be 0.1. For ARMS+PRP, we generated 20 candidate goals for each testing example including the ground-truth goal which corresponds to the ground-truth plan to be recognized. The results are shown in Figure 1.

From Figure 1, we can see that in all three domains, the accu-

racy of our DUP algorithm is generally higher ARMS+PRP, which verifies that our DUP algorithm can indeed capture relations among actions better than the model-based approach ARMS+PRP. The rationale is that we explore global plan information from the plan library to learn a "shallow" model (distributed representations of actions) and use this model with global information to best explain the observed actions. While ARMS+PRP tries to leverage global plan information from the plan library to learn action models and uses the models to recognize observed actions, it enforces itself to extract "exact" models represented by planning models which are often with noise. When feeding those noisy models to PRP, since PRP that uses planning techniques to recognize plans is very sensitive to noise of planning models, the recognition accuracy is lower than DUP, even though ARMS+PRP has more input information (i.e., initial states and candidate goals) than our DUP algorithm.

Looking at the changes of accuracies with respect to the percentage of unobserved actions, we can see that our DUP algorithm performs fairly well even when the percentage of unobserved action reaches 25%. In contrast, ARMS+PRP is sensitive to the percentage of unobserved actions, i.e., the accuracy goes down when more actions are unobserved. This is because the noise of planning models induces more uncertain information, which harms the recognition accuracy, when the percentage of unobserved actions becomes larger. Comparing accuracies of different domains, we can see that our DUP algorithm functions better in the *blocks* domain than the other two domains. This is because the ratio of #word over #vocabulary in the *blocks* domain is much larger than the other two domains, as shown in Table 1. We would conjecture that increasing the ratio could improve the accuracy of DUP. From Figure 1 (as well as Figure 3), we can see that it appears that the accuracy of DUP is not affected by increasing percentages of unobserved actions. The rationale is (1) the percentage of unobserved actions is low, less than 25%, i.e., there is at most one unobserved action over four continuous actions; (2) the window size of context in DUP is set to be 3, which ensures that DUP generally has "stable" context information to estimate the unobserved action when the percentage of unobserved actions is less than 25%, resulting in the stable accuracy in Figure 1 (likewise for Figure 3).

4.1.2 Varying Size of Recommendation Set

We next evaluate the performance of our DUP algorithm with respect to the size of recommendation set S_x. We evaluate the influence of the recommendation set by varying the size from 1 to 10. The size of recommendation set is much smaller than the complete set. For example, the size of complete set in the blocks domain is 1250 (shown in Table 1). It is less than 1% even though we recommend 10 actions for each unobserved action. We set the context window c used in Equation (1) to be three, the percentage of unobserved actions to be 0.25, and the learning constant δ in Equation (6) to be 0.1. For ARMS+PRP, the number of candidate goals for each testing example is set to 20. ARMS+PRP aims to recognize plans that are optimal with respect to the cost of actions. We relax ARMS+PRP to output $|S_x|$ optimal plans, some of which might be suboptimal. The results are shown in Figure 2.

From Figure 2, we find that accuracies of the three approaches generally become larger when the size of the recommended set increases in all three domains. This is consistent with our intuition, since the larger the recommended set is, the higher the possibility for the *truth* action to be in the recommended set. We can also see that the accuracy of our DUP algorithm are generally larger than ARMS+PRP in all three domains, which verifies that our DUP algorithm can indeed better capture relations among actions and thus recognize unobserved actions better than the model-learning based approach ARMS+PRP. The reason is similar to the one given for Figure 1 in the previous section. That is, the "shallow" model learnt by our DUP algorithm is better for recognizing plans than both the "exact" planning model learnt by ARMS for recognizing plans with planning techniques. Furthermore, the advantage of DUP becomes even larger when the size of recommended action set increases, which suggests our vector representation based learning approach can better capture action relations when the size of recommended action set is larger. The possibility of actions correctly recognized by DUP becomes much larger than ARMS+PRP when the size of recommendations increases.

4.2 Comparison between DUP and MatchPlan

In this experiment we compare DUP to MatchPlan which is built based on the idea of [9]. Likewise we varied the percentage of unobserved actions and the size of recommended actions to see the change of accuracies of both algorithms. The results are exhibited below.

4.2.1 Varying Percentage of Unobserved Actions

To compare our DUP algorithm with MatchPlan with respect to different percentage of unobserved actions, we set the window of training context c in Equation (1) of DUP to be three, the number of iterations in Algorithm 1 to be 1500, the size of recommendations to be ten, and the learning constant δ in Equation (6) to be 0.1. To make fair the comparison (with MatchPlan), we set the matching window of MatchPlan to be three, the same as the training context c of DUP, when searching plans from plan libraries $\mathcal{L}$. In other words, to estimate an unobserved action $\bar{a}_x$ in $\mathcal{O}$, MatchPlan matches previous three actions and subsequent three actions of $\bar{a}_x$ to plans in $\mathcal{L}$, and recommends ten actions with the maximal number of matched actions, considering observed actions in the context of $\bar{a}_x$ and actions in $\mathcal{L}$ as a successful matching. The results are shown in Figure 3.

From Figure 3, we find that the accuracy of DUP is much better than MatchPlan, which indicates that our DUP algorithm can better learn knowledge from plan libraries than the local matching approach MatchPlan. This is because we take advantage of global plan information of the plan library when learning the "shallow" model, i.e., distributed representations of actions, and the model with global information can best explain the observed actions. In contrast, MatchPlan just utilizes local plan information when matching the observed actions to the plan library, which results in lower accuracies. Looking at all three different domains, we can see that both algorithms perform the best in the *blocks* domain. The reason is similar to the one provided in the last subsection (for Figure 1), i.e., the number of words over the number of vocabulary in the *blocks* domain is relatively larger than the other two domains, which gives us the hint that it is possible to improve accuracies by increasing the ratio of the number of words over the number of vocabularies.

4.2.2 Varying Size of Recommendation Set

Likewise, we also would like to evaluate the change of accuracies when increasing the size of recommended actions. We used the same experimental setting as done by previous subsection. That is, we set the window of training context c of DUP to be three, the learning constant δ to be 0.1, the number of iterations in Algorithm 1 to be 1500, the matching window of MatchPlan to be three. In addition, we fix the percentage of unobserved actions to be 0.25. The results are shown in Figure 4.

We can observe that the accuracy of our DUP algorithm are generally larger than MatchPlan in all three domains in Figure 4,

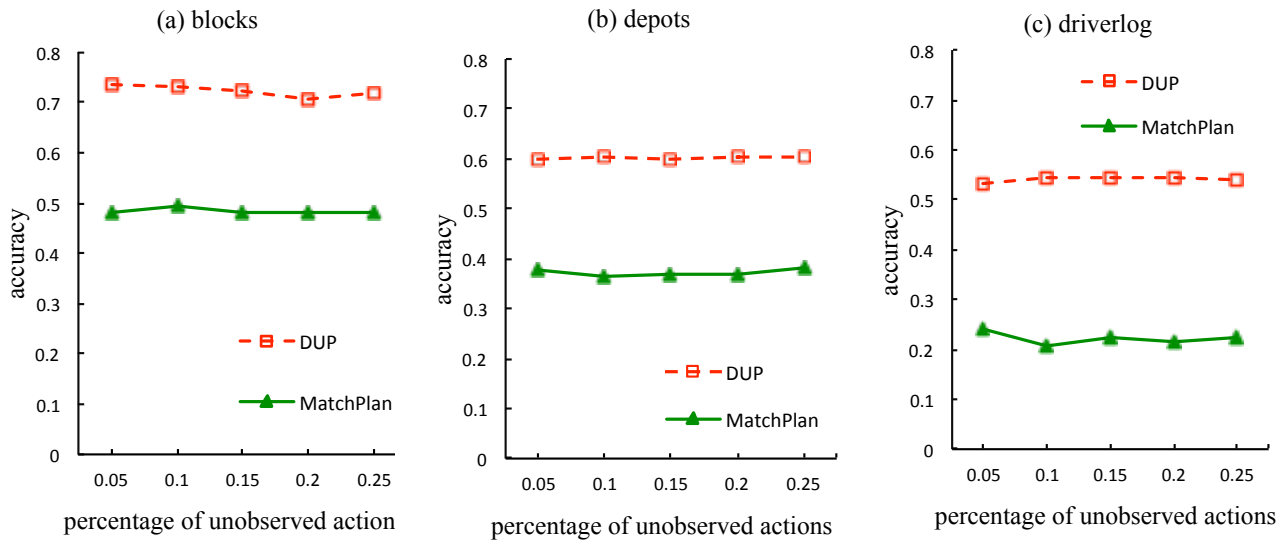

Figure 3: Accuracies of `DUP` and `MatchPlan` with respect to different percentage of unobserved actions

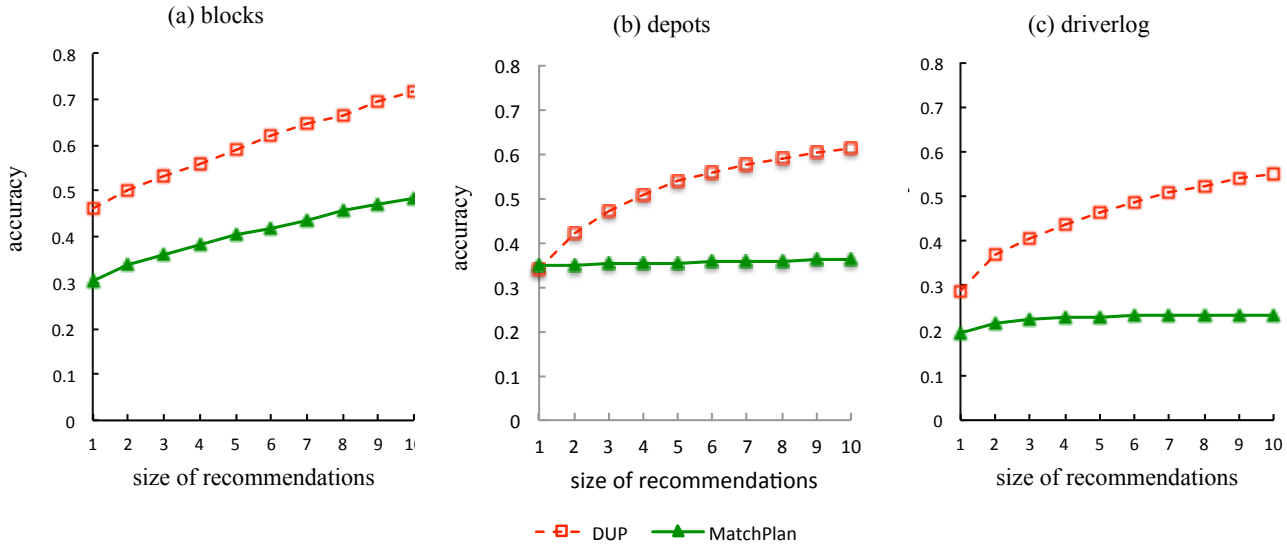

Figure 4: Accuracies of `DUP` and `MatchPlan` with respect to different size of recommendations

which suggests that our `DUP` algorithm can indeed better capture relations among actions and thus recognize unobserved actions better than the matching approach `MatchPlan`. The reason behind this is similar to previous experiments, i.e., the global information captured from plan libraries by `DUP` can indeed better improve accuracies than local information exploited by `MatchPlan`. In addition, looking at the trends of the curves of both `DUP` and `MatchPlan`, we can see the performance of `DUP` becomes much better than `MatchPlan` when the size of recommendations increases. This indicates the influence of global information becomes much larger when the size of recommendations increasing. In other words, larger size of recommendations provides better chance for "shallow" models learnt by `DUP` to perform better.

4.2.3 Varying Size of Training Set

To see the effect of size of training set, we ran both `DUP` and `MatchPlan` with different size of training set. We used the same setting as done by last subsection except fixing the size of recommendations to be 10, when running both algorithms. We varied the size of training set from 2500 to 4500. The results are shown in Figure 5.

We observed that accuracies of both `DUP` and `MatchPlan` generally become higher when the size of training set increases. This is consistent with our intuition, since the larger the size of training set is, the richer the information is available for improving the accuracies. Comparing the curves of `DUP` and `MatchPlan`, we can see that `DUP` performs much better than `MatchPlan`. This further

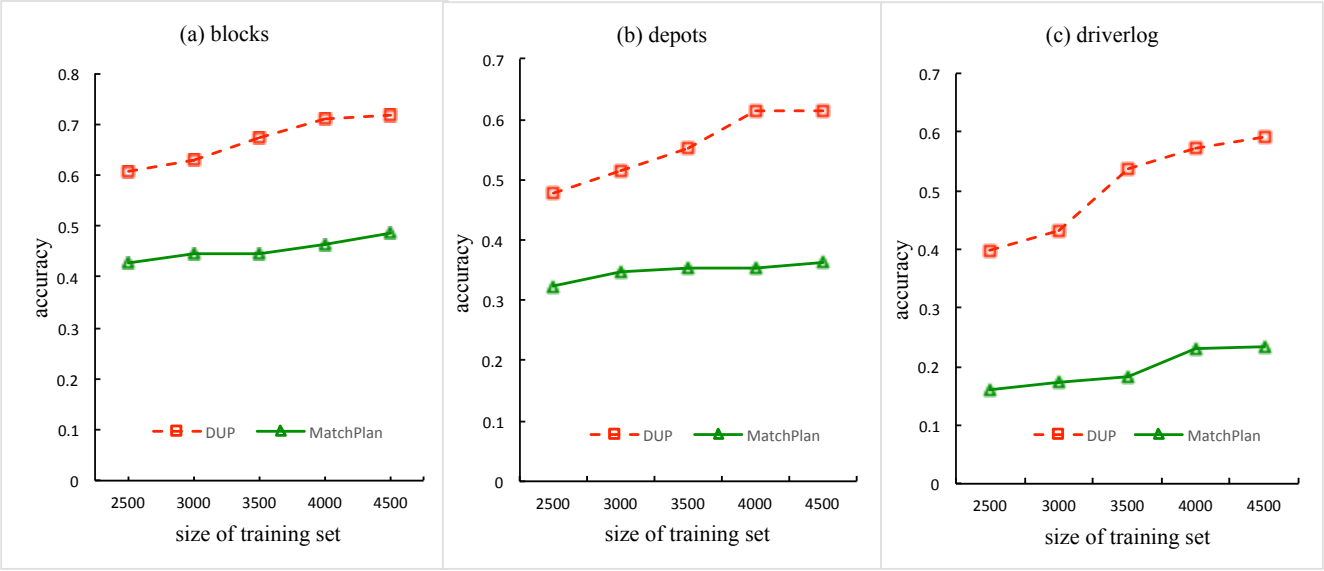

Figure 5: Accuracies of `DUP` and `MatchPlan` with respect to different size of training set

verifies the benefit of exploiting global information of plan libraries when learning the shallow models as done by `DUP`.

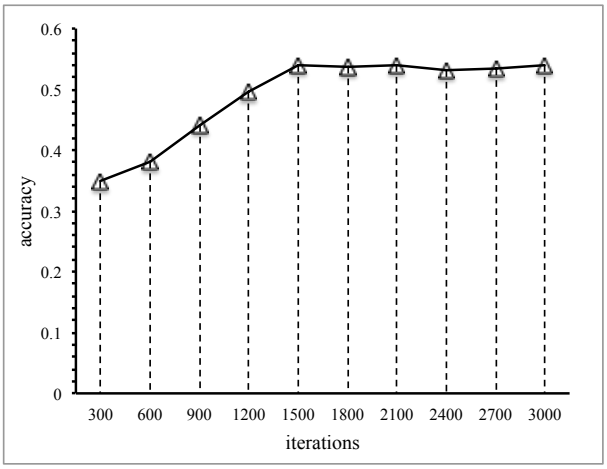

Figure 6: Accuracy with respect to different number of iterations in the blocks domain

4.3 Accuracy w.r.t. Iterations

In the previous experiments, we set the number of iterations in Algorithm 1 to be 1500. In this experiment, we would like to see the influence of iterations of our `DUP` algorithm when running the EM-style procedure. We changed the number of iterations from 300 to 3000 to see the trend of accuracy. We exhibit the experimental result in the *blocks* domain (the results of the other two domains are similar) in Figure 6.

From Figure 6, we can see the accuracy becomes higher at the beginning and stays flat when reaching the size of 1500. This exhibits that the EM procedure converges and has stable accuracies after the iteration reaches 1500. Similar results can also be found in the other two domains.

5. RELATED WORK

Our work is related to planning with incomplete domain models (or model-lite planning [12, 29]). Figure 7 shows the schematic view of incomplete models and their relationships in the spectrum of incompleteness. In a full model, we know exactly the dynamics of the model (i.e., state transitions). Approximate models are the closest to full models and their representations are similar except that there can be incomplete knowledge of action descriptions. To enable approximate planners to perform more (e.g., providing robust plans), planners are assumed to have access to additional knowledge circumscribing the incompleteness [25]. Partial models are one level further down the line in terms of the degree of incompleteness. While approximate models can encode incompleteness in the precondition/effect descriptions of the individual actions, partial models can completely abstract portions of a plan without providing details for them. In such cases, even though providing complete plans is infeasible, partial models can provide "planning guidance" for agents [27]. Shallow models are essentially just a step above having no planning model. They provide interesting contrasts to the standard precondition and effect based action models used in automated planning community. Our work in this paper belongs to the class of shallow models. In developing shallow models, we are interested in planning technology that helps humans develop plans, even in the absence of any structured models or plan traces. In such cases, the best that we can hope for is to learn local structures of the planning model to provide planning support, similar to providing spell-check in writing. While some work in web-service composition (c.f. [7]) did focus on this type of planning support, they were hobbled by being limited to simple input/output type comparison. In contrast, we expect shallow models to be useful in "critiquing" the plans being generated by the humans (e.g. detecting that an action introduced by the human is not consistent with the model), and "explaining/justifying" the suggestions generated by humans.

Our work is also related to plan recognition. Kautz and Allen proposed an approach to recognizing plans based on parsing observed actions as sequences of subactions and essentially model this knowledge as a context-free rule in an "action grammar" [14].

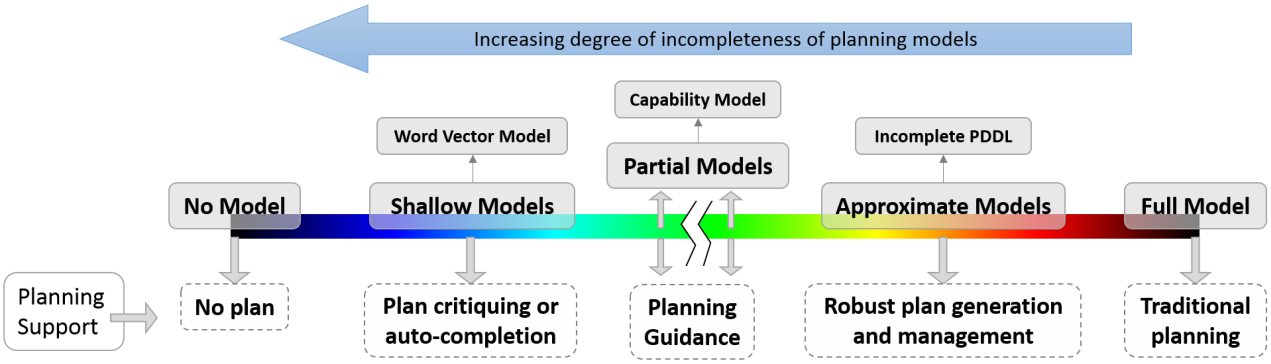

Figure 7: Schematic view of incomplete models and their relationships in the spectrum of incompleteness

All actions, plans are uniformly referred to as goals, and a recognizer's knowledge is represented by a set of first-order statements called event hierarchy encoded in first-order logic, which defines abstraction, decomposition and functional relationships between types of events. Lesh and Etzioni further presented methods in scaling up activity recognition to scale up his work computationally [16]. They automatically constructed plan-library from domain primitives, which was different from [14] where the plan library was explicitly represented. In these approaches, the problem of combinatorial explosion of plan execution models impedes its application to real-world domains. Kabanza and Filion [10] proposed an anytime plan recognition algorithm to reduce the number of generated plan execution models based on weighted model counting. These approaches are, however, difficult to represent uncertainty. They offer no mechanism for preferring one consistent approach to another and incapable of deciding whether one particular plan is more likely than another, as long as both of them can be consistent enough to explain the actions observed. Although we exploit a library of plans in our DUP approach, we aim to learning shallow models and utilize the shallow models to recognize plans that are not necessarily in the plan library, which is different from previous approaches that assume the plans to be recognized are from the plan library.

Instead of using a library of plans, Ramirez and Geffner [22] proposed an approach to solving the plan recognition problem using slightly modified planning algorithms, assuming the action models were given as input (note that action models can be created by experts or learnt by previous systems [26, 30]). Except previous work [14, 4, 8, 22] on the plan recognition problem presented in the introduction section, Saria and Mahadevan presented a hierarchical multi-agent markov processes as a framework for hierarchical probabilistic plan recognition in cooperative multi-agent systems [24]. Amir and Gal addressed a plan recognition approach to recognizing student behaviors using virtual science laboratories [2]. Ramirez and Geffner exploited off-the-shelf classical planners to recognize probabilistic plans [23]. Different from those approaches, we do not require any domain model knowledge provided as input. Instead, we automatically learn shallow domain models from previous plan cases for recognizing unknown plans that may not be identical to previous cases.

6. CONCLUSION AND DISCUSSION

In this paper we present a novel plan recognition approach DUP based on vector representation of actions. We first learn the vector representations of actions from plan libraries using the Skip-gram model which has been demonstrated to be effective. We then discover unobserved actions with the vector representations by repeatedly sampling actions and optimizing the probability of potential plans to be recognized. We also empirically exhibit the effectiveness of our approach.

While we focused on a one-shot recognition task in this paper, in practice, human-in-the-loop planning will consist of multiple iterations, with DUP recognizing the plan and suggesting action addition alternatives; the human making a selection and revising the plan. The aim is to provide a form of flexible plan completion tool, akin to auto-completers for search engine queries. To do this efficiently, we need to make the DUP recognition algorithm "incremental."

The word-vector based domain model we developed in this paper provides interesting contrasts to the standard precondition and effect based action models used in automated planning community. One of our future aims is to provide a more systematic comparison of the tradeoffs offered by these models. Although we have focused on the "plan recognition" aspects of this model until now, and assumed that "planning support" will be limited to suggesting potential actions to the humans. In future, we will also consider "critiquing" the plans being generated by the humans (e.g. detecting that an action introduced by the human is not consistent with the model learned by DUP), and "explaining/justifying" the suggestions generated by humans. Here, we cannot expect causal explanations of the sorts that can be generated with the help of complete action models (e.g. [20]), and will have to develop justifications analogous to those used in recommendation systems.

Another potential application for this type of distributed action representations proposed in this paper is social media analysis. In particular, work such as [15] shows that identification of action-outcome relationships can significantly improve the analysis of social media threads. The challenge of course is that such action-outcome models have to be learned from raw and noisy social media text containing mere fragments of plans. We believe that action vector models of the type we proposed in this paper provide a promising way of handling this challenge.

Acknowledgements

Tian and Zhuo thank National Natural Science Foundation of China (No. 61309011) and Key Lab of Machine Intelligence and Advanced Computing, Ministry of Education of China 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

[1] S. V. Albrecht and S. Ramamoorthy. Are you doing what I think you are doing? criticising uncertain agent models. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, pages 52–61, 2015.

[2] O. Amir and Y. K. Gal. Plan recognition in virtual laboratories. In *Proceedings of IJCAI*, pages 2392–2397, 2011.

[3] S. Branavan, N. Kushman, T. Lei, and R. Barzilay. Learning high-level planning from text. In *Proceedings of ACL-12*, 2012.

[4] H. H. Bui. A general model for online probabilistic plan recognition. In *Proceedings of IJCAI*, pages 1309–1318, 2003.

[5] P. R. Cohen, E. C. Kaiser, M. C. Buchanan, S. Lind, M. J. Corrigan, and R. M. Wesson. Sketch-thru-plan: a multimodal interface for command and control. *Commun. ACM*, 58(4):56–65, 2015.

[6] X. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Simlarity search for web services. In *(e)Proceedings of the Thirtieth International Conference on Very Large Data Bases*, pages 372–383, 2004.

[7] X. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Simlarity search for web services. In *Proceedings of VLDB*, pages 372–383, 2004.

[8] C. W. Geib and R. P. Goldman. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence*, 173(11):1101–1132, 2009.

[9] C. W. Geib and M. Steedman. On natural language processing and plan recognition. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 1612–1617, 2007.

[10] F. Kabanza, J. Filion, A. R. Benaskeur, and H. Irandoust. Controlling the hypothesis space in probabilistic plan recognition. In *IJCAI*, 2013.

[11] S. Kambhampati. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 1601–1605, 2007.

[12] S. Kambhampati. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain theories. In *Proceedings of AAAI*, pages 1601–1605, 2007.

[13] S. Kambhampati and K. Talamadupula. Human-in-the-loop planning and decision support. 2015. rakaposhi.eas.asu.edu/hilp-tutorial.

[14] H. A. Kautz and J. F. Allen. Generalized plan recognition. In *Proceedings of AAAI*, pages 32–37, 1986.

[15] E. Kıcıman and M. Richardson. Towards decision support and goal achievement: Identifying action-outcome relationships from social media. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 547–556. ACM, 2015.

[16] N. Lesh and O. Etzioni. A sound and fast goal recognizer. In *IJCAI*, pages 1704–1710, 1995.

[17] L. Manikonda, T. Chakraborti, S. De, K. Talamadupula, and S. Kambhampati. AI-MIX: using automated planning to steer human workers towards better crowdsourced plans. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 3004–3009, 2014.

[18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[19] A. Y. Ng, H. J. Kim, M. I. Jordan, and S. Sastry. Autonomous helicopter flight via reinforcement learning. In *Proceedings of NIPS-03*, 2003.

[20] C. J. Petrie. Constrained decision revision. In *Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, CA, July 12-16, 1992.*, pages 393–400, 1992.

[21] M. Ramírez and H. Geffner. Plan recognition as planning. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 1778–1783, 2009.

[22] M. Ramirez and H. Geffner. Plan recognition as planning. In *Proceedings of IJCAI*, pages 1778–1783, 2009.

[23] M. Ramirez and H. Geffner. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of AAAI*, pages 1121–1126, 2010.

[24] S. Saria and S. Mahadevan. Probabilistic plan recognitionin multiagent systems. In *Proceedings of AAAI*, 2004.

[25] M. D. Tuan Nguyen, Subbarao Kambhampati. Synthesizing robust plans under incomplete domain models. In *Proc. AAAI Workshop on Generalized Planning*, 2011.

[26] Q. Yang, K. Wu, and Y. Jiang. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence Journal*, 171:107–143, February 2007.

[27] Y. Zhang, S. Sreedharan, and S. Kambhampati. Capability models and their applications in planning. In *Proceedings of AAMAS*, pages 1151–1159, 2015.

[28] H. H. Zhuo and L. Li. Multi-agent plan recognition with partial team traces and plan libraries. In *Proceedings of IJCAI*, pages 484–489, 2011.

[29] H. H. Zhuo, T. A. Nguyen, and S. Kambhampati. Model-lite case-based planning. In *AAAI*, 2013.

[30] H. H. Zhuo, Q. Yang, D. H. Hu, and L. Li. Learning complex action models with quantifiers and implications. *Artificial Intelligence*, 174(18):1540–1569, 2010.

[31] H. H. Zhuo, Q. Yang, and S. Kambhampati. Action-model based multi-agent plan recognition. In *Proceedings of NIPS*, pages 377–385, 2012.