Discovering Underlying Plans Based on Shallow Models

HANKZ HANKUI ZHUO, Sun Yat-Sen University
YANTIAN ZHA and SUBBARAO KAMBHAMPATI, Arizona State University
XIN TIAN, Sun Yat-Sen University

Plan recognition aims to discover target plans (i.e., sequences of actions) behind observed actions, with history plan libraries or action 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 action models to best explain the observed actions, assuming that complete action models are available. In real-world applications, however, target plans are often not from plan libraries, and complete action models are often not available, since building complete sets of plans and complete action models are often difficult or expensive. In this article, 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. Specifically, we propose two approaches, `DUP` and `RNNPlanner`, to discover target plans based on vector representations of actions. `DUP` explores the EM-style (Expectation Maximization) framework to capture local contexts of actions and discover target plans by optimizing the probability of target plans, while `RNNPlanner` aims to leverage long-short term contexts of actions based on RNNs (Recurrent Neural Networks) framework to help recognize target plans. In the experiments, we empirically show that our approaches are capable of discovering underlying plans that are not from plan libraries without requiring action models provided. We demonstrate the effectiveness of our approaches by comparing its performance to traditional plan recognition approaches in three planning domains. We also compare `DUP` and `RNNPlanner` to see their advantages and disadvantages.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; **Planning and scheduling**;

Additional Key Words and Phrases: Plan recognition, shallow model, action representation, recurrent neural networks

ACM Reference format:
Hankz Hankui Zhuo, Yantian Zha, Subbarao Kambhampati, and Xin Tian. 2020. Discovering Underlying Plans Based on Shallow Models. *ACM Trans. Intell. Syst. Technol.* 11, 2, Article 18 (January 2020), 30 pages.
https://doi.org/10.1145/3368270

Zhuo thanks the support of the National Natural Science Foundation of China (U1611262), Guangdong Natural Science Fund for Distinguished Young Scholars (2017A030306028), Guangdong special branch plans young talent with scientific and technological innovation, Pearl River Science and Technology New Star of Guangzhou, and Guangdong Province Key Laboratory of Big Data Analysis and Processing for the support of this research. Kambhampati's research is supported in part by the ONR grants N00014-16-1-2892, N00014-18-1-2442, N00014-18-1-2840, the AFOSR grant FA9550-18-1-0067, and the NASA grant NNX17AD06G.
Authors' addresses: H. H. Zhuo (corresponding author) and X. Tian, School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, China and Key Laboratory of Machine Intelligence and Advanced Computing (Sun Yat-Sen University) Ministry of Education, China; emails: zhuohank@mail.sysu.edu.cn, tianxin1860@gmail.com; Y. Zha and S. Kambhampati, School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ 85281 USA; emails: {yzha3, rao}@asu.edu.

2157-6904/2020/01-ART18 $15.00
https://doi.org/10.1145/3368270

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 (cf. References [17, 18, 41]). An important aspect of such a support [33] is recognizing what plans the human-in-the-loop is making and provide appropriate suggestions about their next actions [2]. Additionally, discovering missing actions executed in the past by other agents based on "some" historical observations would be helpful for humans to make better decisions to better cooperate with others.

Although there is a lot of work on plan recognition, much of it has traditionally depended on the availability of complete action models [55, 74, 80]. As has been argued elsewhere [33], 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 cooperative 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 (cf. Reference [31]), which can be easily learned automatically. Compared to learning action models [65, 76, 78, 79] ("complex" models, correspondingly) of the domain from limited training data, learning shallow models can avoid the overfitting issue. Learning shallow models in this work is different from previous HMM-based plan recognition approaches [11, 22] that were based on observable signals. We aim to learn shallow models only based on a symbolic plan library. Our usage of "shallow models" follows the description given by Reference [31], where they are defined as models that are mostly useful for critiquing rather than creations of plans.

There has been very little work on learning such shallow models to support human-in-the-loop planning. For example, the work on Woogle system [18] aimed to provide support to humans in web-service composition. That work, however, relied on a very primitive understanding of the actions (web services, in their case) that consisted merely of learning the input/output types of individual services. Another work on IBM Scenario Planning Advisor [60] generates lists of candidate observations corresponding to the detected risk drivers. That work, however, relied on aggregating relevant news from the Web and social media. Other than supporting human-in-the-loop planning, work such as Reference [8] considers shallow models to be represented by deep latent space based on raw information such as images describing states between actions. This is different from our plan recognition problem, which aims to build shallow models based on symbolic action sequences without any state-related information available. In this article, we focus on learning more informative models that can help recognize the plans under construction by humans, and provide active support by suggesting relevant actions without any information about states between actions.

Inspired by the effectiveness of distributed representations of words [43] in natural language, where each word is represented by a vector and semantic relationships among words can be characterized by their corresponding vectors, we found that actions can be viewed as words and plans can be seen as sentences (and plan libraries can be viewed as corpora), suggesting actions can be represented by vectors. As a result, similar to calculating probability of sentences based on vector representations of words, we can calculate the probability of underlying candidate plans based on vector representations of actions. Based on the above-mentioned discovery, we propose two approaches to learning informative models: `DUP`, standing for **D**iscovering **U**nderlying **P**lans based on action-vector representations, and `RNNPlanner`, standing for **R**ecurrent **N**eural **N**etwork based **Planner**. The framework of `DUP` and `RNNPlanner` is shown in Figure 1, where we take as input a set of plans (or a *plan library*) and learn the distributed representations of actions (namely, *action vectors*). After that, our `DUP` approach exploits an EM-style (Expectation Maximization) framework to discover underlying plans based on the learnt action vectors, while our `RNNPlanner` approach exploits an RNN-style framework to generate plans to best explain observations (i.e., discover

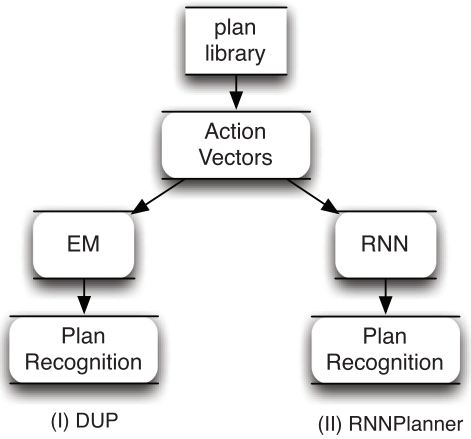

Fig. 1. The framework of our shallow models DUP and RNNPlanner.

underlying plans behind the observed actions) based on the learnt action vectors. In DUP, we consider local contexts (with a limited window size) of actions being recognized, while in RNNPlanner, we explore the potential influence from long- and short-term actions, which can be modelled by RNN, to help recognize unknown actions. In DUP, we calculate the post probability of an action given its local context (preset by a specific window size) and thereafter the probability of a plan by the product of all probabilities of actions in the plan. In other words, in the EM-style framework of DUP, we consider local contexts of actions. However, RNNPlanner aims to remember actions that are "far" from and have strong impacts on the target action. In other words, RNNPlanner is able to capture long- and short-term actions that "influence" the occurrence of the target action. In this article, we investigate the advantage and disadvantage of utilizing local contexts, i.e., DUP, and long- and short-term contexts, i.e., RNNPlanner, respectively.

In summary, the contributions of the article are shown below.

(1) In Reference [62], we presented a version of DUP. In this article, we extend Reference [62] with more details to elaborate the approach.
(2) We propose a novel model RNNPlanner based on RNN to explore the influence of actions from long- and short-term contexts.
(3) We compare RNNPlanner to DUP to exhibit the advantages and disadvantages of leveraging information from long- and short-term contexts.

In the following sections, we first formulate our plan recognition problem and then address the details of our approaches DUP and RNNPlanner. After that, we 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 previous plan recognition techniques, including the one that uses plan traces to learn the STRIPS-style action models, and use the learned model to support plan recognition. We also compare RNNPlanner with DUP to see the the advantage and disadvantage of leveraging long- and short-term contexts of actions in different scenarios. We finally review previous approaches related to our work and conclude our article with further work.

2 PROBLEM FORMULATION

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 $O = \langle o_1, o_2, \ldots, o_M \rangle$, 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 anymore.

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 of the plan or in the middle. Our aim is to suggest, for each of the k unobserved actions, l possible choices from which the user can select the action. (Note that we would like to keep l small, ideally close to 1, so as not to overwhelm users.) 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 l actions.

Specifically, our recognition problem can be represented by a triple $\mathfrak{R} = (\mathcal{L}, O, \mathcal{A})$. The solution to $\mathfrak{R}$ is to discover the unknown plan $\tilde{p}$, which is a plan with unknown observations that best explains O given $\mathcal{L}$ and $\mathcal{A}$. We have the following assumptions **A1–A3**:

- A1: The length of the underlying plan to be discovered is known, which releases us from searching unlimited length of plans.
- A2: The positions of missing actions in the underlying plan are known in advance, which releases us from searching missing actions in between observed actions.
- A3: All actions observed are assumed to be correct, which indicates there is no need to criticize or rectify the observed actions.

Note that assumptions **A1** and **A2** can be removed by leveraging auxiliary knowledge, such as domain models, as done by previous approaches [20, 55, 61]. Building such auxiliary knowledge often requires a lot of human efforts, thus, we assume we do not have that knowledge by considering assumptions **A1** and **A2**.

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 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., References [55, 75, 80]. While these

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

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 the same as our problem we aim to solve.

3 LEARNING THE DISTRIBUTED REPRESENTATIONS OF ACTIONS

To discover the unobserved actions, we aim to estimate the probability of candidate actions that can be the unobserved actions. To do this, inspired by distributed representations of words [43], we aim to learn the distributed representations of actions and exploit the representations to compute the probability of candidate actions (together with observed 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. Given a plan corpus, we exploited the off-the-shelf software, word2vec [43], for learning vector representations for actions.

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 C, composed of a sequence of training words $\langle w_1, w_2, \ldots, w_T \rangle$, where $T = |C|$, the Skip-gram model maximizes the average log probability

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

where c is the size of the training window or context. Note that "empty" words are added at the beginning of the word sequence when $t + j$ is less than 1, and in the end of the word sequence when $t + j$ is larger than T.

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 representing the relative probabilities of its child nodes [43, 45]. Each word v must be represented by a bit vector $(b_1(v), \ldots, b_m(v))$. This can be achieved by building a binary hierarchical clustering of words. For example, $b_1(v) = 1$ indicates that v belongs to the top-level group 1 and $b_2(v) = 0$ indicates that it belongs to the sub-group 0 of that top-level group. The next-word conditional probability can thus be represented and computed by:

$$P(v|w_{t-1}, \ldots, w_{t-n+1}) = \prod_{j=1}^{m} P(b_j(v)|b_1(v), \ldots, b_{j-1}(v), w_{t-1}, \ldots, w_{t-n+1}).$$

This can be interpreted as a series of binary stochastic decisions associated with nodes of a binary tree. Each node is indexed by a bit vector corresponding to the path from the root to the node (append 1 or 0 according to whether the left or right branch of a decision node is followed). Each leaf corresponds to a word. For each leaf node, there is a 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

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

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.

4 OUR DUP ALGORITHM

Our DUP approach to the recognition problem $\mathfrak{R}$ 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.

4.1 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.,

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

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 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 O, i.e., the number of ϕs in O is X, we denote these unobserved actions by $\bar{a}_1, \ldots, \bar{a}_x, \ldots, \bar{a}_X$, where the indices indicate the order they appear in 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 $\bar{\Gamma}_{o,x} \ge 0$ 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 O, for all $o \in \bar{\mathcal{A}}$; otherwise, $\Gamma_{o,y} = 1$ and $\Gamma_{o',y} = 0$ for all $o' \in \bar{\mathcal{A}} \wedge o' \ne o$. 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,

$$\begin{aligned} p(w_{k+j} | w_k) = & \prod_{i=1}^{L(w_{k+j})-1} \Big\{ \sigma(\mathbb{I}(n(w_{k+j}, i+1) \\ & = child(n(w_{k+j}, i))) \cdot a v_{n(w_{k+j}, i)} \cdot b v_{w_k}) \Big\}, \end{aligned} \tag{4}$$

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 assume $X \sim Multinomial(\Gamma_{\cdot,x})$, i.e., $p(X = o) = \Gamma_{o,x}$, where $\Gamma_{o,x} \ge 0$ and $\sum_{a \in \bar{\mathcal{A}}} \eta^a = 1$. $P(\tilde{p}|\Gamma) = \prod_x \Gamma_{o,x}$.

We redefine the objective function as shown below,

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

where $p(w_{k+j}|w_k)$ is defined by Equation (4). The gradient of Equation (5) is shown below,

$$\frac{\partial \mathcal{F}}{\partial \Gamma_{o,x}} = \frac{4c(L(o) - 1)}{\Gamma_{o,x}}. \tag{6}$$

The only parameters needed to be updated are Γ, which can be easily done by gradient descent, as shown below,

$$\Gamma_{o,x} = \Gamma_{o,x} + \delta \frac{\partial \mathcal{F}}{\partial \Gamma_{o,x}}, \tag{7}$$

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

With Equation (7), we can design an EM algorithm by repeatedly sampling an unknown plan according to Γ and updating Γ based on Equation (7) until reaching convergence (e.g., a constant number of repetitions is reached).

4.2 Overview of Our DUP Approach

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 O; and, otherwise, $\Gamma_{o,k} = 1$ and $\Gamma_{o',k} = 0$ for all $o' \in \bar{\mathcal{A}} \wedge o' \ne o$. 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 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 $\mathcal{L}$, an observation O
Output: plan $\tilde{p}$

1: learn vector representation of actions based on $\mathcal{L}$
2: initialize $\Gamma_{o,k}$ with $1/M$ for each $o \in \bar{\mathcal{A}}$ and each k that is an unobserved action index
3: **while** the maximal number of repetitions Λ is not reached **do**
4: sample unobserved actions in O based on Γ
5: for each unobserved action x, update $\Gamma_{\cdot,x}$ based on Equation (7)
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 that has been successfully applied to complex problems, e.g., robot control [46], natural language processing [10]. Our formulation is unique in how it recognizes plans, in comparison to the existing methods in the planning community. The first step of Algorithm 1 is $O(Iters \times T_{max} \times |\mathcal{L}| \times WinSize)$, where $Iters$ is the maximal number of iterations for vector representations of actions, and T_{max} is the

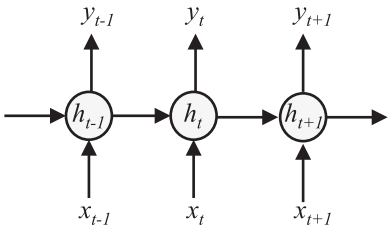

Fig. 2. The framework of Recurrent Neural Networks.

maximal length of plans in $\mathcal{L}$ and $|\mathcal{L}|$ is the number of plans in $\mathcal{L}$, and $WinSize$ is the size of the window ($WinSize = 2c + 1$). The time cost of Step 2 is $O(|\mathcal{A}| \times unobserved(O))$, and the time cost of Steps 2 to 7 is $O(\Lambda \times |\mathcal{A}| \times unobserved(O))$, where $unobserved(O)$ is the number of unobserved actions in O. In general, the time cost of the first step is much higher than other steps, i.e., the time cost of Algorithm 1 is $O(Iters \times T_{max} \times |\mathcal{L}| \times WinSize)$.

5 OUR RNNPLANNER APPROACH

In the EM-style framework, we aim to randomly sample actions of unobserved actions and then estimate the probability of them together with observed actions. This framework is capable of capturing local context of unobserved actions well (i.e, *a posteriori* probability of unobserved actions is conditionally dependent on preceding and succeeding actions with a window). In this section, we also would like to study the utility of long-term context for predicting unobserved actions based on observed actions and plan library. To do this, we explore a well-known model, RNNs, specifically Long Short-Term Memory networks (LSTMs), which have been demonstrated effective on predicting future signals [49] by remembering long-term information. We will first introduce the RNN architecture and then introduce our `RNNPlanner` model.

5.1 The RNN Model

We consider a specific type of RNN that generates an output sequence with the same length of an input sequence. Note that in the Natural Language Processing (NLP) community, various RNN models, such as deep RNNs [49], Bi-LSTM-CRF [4], and so on, were designed to suit different complicated NLP tasks. The more complex the model is, the more data are required to build the model. Compared to NLP tasks, our plan recognition task is relatively not that complicated (plans in the plan library are formally represented by action sequences, while natural language data in NLP tasks are often unstructured and ambiguous). In addition, our data are relatively smaller than that used in many RNN models in complicated NLP tasks. To avoid overfitting, we chose a basic RNN model (with a relatively small number of parameters). Given an input sequence $\mathbf{x}$, an RNN model could predict an output sequence $\mathbf{y}$, in which output y_t at each step depends on the input x_t at that step and the hidden state at the previous step. If we unroll this RNN cell along T steps, then it could accept an input sequence $\mathbf{x} = (x_1 \ldots, x_T)$ and compute a sequence of hidden states $\mathbf{h} = (h_1, h_2, \ldots, h_T)$, as shown in Figure 2. For each of these hidden states h_t ($1 \leq t \leq T$), it contributes to predicting the next step output y_{t+1}, and thus RNN computes an output vector sequence $\mathbf{y} = (y_1, \ldots, y_T)$ by iterating the following equations from $t = 1$ to T:

$$h_t = \mathcal{H}(W_{xh}x_t + W_{hh}h_{t-1} + b_h), \tag{8}$$

$$y_t = W_{hy}h_t + b_y, \tag{9}$$

where the W terms denote weight matrices, i.e., W_{xh} is the input-hidden weight matrix, W_{hh} is the hidden-hidden weight matrix, and W_{hy} is the hidden-output matrix. The b terms denote bias

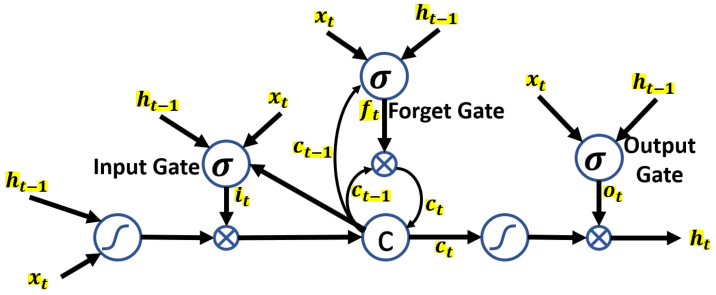

Fig. 3. The framework of the Long Short-Term Memory (LSTM) cell.

vectors, i.e., b_h is hidden bias vector and b_y is the output bias vector. $\mathcal{H}$ is the hidden layer function, which is usually an elementwise application of a sigmoid function. The RNN could also be utilized to directly generate, in principle, infinitely long future outputs, given a single input x_t. The sequence of future outputs could be generated by directly feeding the output y_t at a step t, to the input x_{t+1} at the next step $t+1$, by assuming what it predicts is reliable (i.e., $y_t = x_{t+1}$). As for training the RNN as a sequence generation model, we could utilize y_t to parameterize a predictive distribution $P(x_{t+1}|y_t)$ over all of the possible next inputs x_{t+1}, by minimizing the loss:

$$\mathcal{L}(\mathbf{x}) = -\sum_{t=1}^{T} logP(x_{t+1}|y_t), \tag{10}$$

where T is the number of steps of an observed plan trace, x_{t+1} is the observed action at step $t+1$, and y_t is the output at step t as well as the prediction of what would happen at step $t+1$. To estimate y_t based on $x_1, \ldots, x_t$, we exploit the LSTM model [24, 58], which has been demonstrated effective on generating sequences, to leverage long-term information prior to x_t and predict y_t based on current input x_t. We can thus rewrite Equation (10) as:

$$\mathcal{L}(\mathbf{x};\theta) = -\sum_{t=1}^{T} logP(x_{t+1}|y_t)\text{LSTM}(y_t|x_{1:t};\theta), \tag{11}$$

where $\text{LSTM}(y_t|x_{1:t};\theta)$ indicates the LSTM model estimates y_t based on current input x_t and memories of previous input prior to x_t. θ are parameters including W_{xh}, W_{hh}, W_{hy}, b_h, and b_y.

The framework of LSTM [24, 58] is shown in Figure 3, where x_t is the tth input, h_t is the tth hidden state. i_t, f_t, o_t, and c_t are the tth *input gate*, *forget gate*, *output gate*, *cell*, and *cell input* activation vectors, respectively, whose dimensions are the same as the hidden vector h_t. LSTM is implemented by the following functions:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i), \tag{12}$$
$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f), \tag{13}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c), \tag{14}$$
$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o), \tag{15}$$
$$h_t = o_t \circ \tanh(c_t), \tag{16}$$

where $\circ$ indicates the Hadamard product, σ is the logistic sigmoid function, W_{xi} is an input-input gate matrix, W_{hi} is a hidden-input gate matrix, W_{ci} is a cell-input gate matrix, W_{xf} is an input-forget gate matrix, W_{hf} is a hidden-forget gate matrix, W_{cf} is a cell-forget gate matrix, W_{xc} is an input-cell gate matrix, W_{hc} is a hidden-cell gate matrix, W_{xo} is an input-output gate matrix, W_{ho}

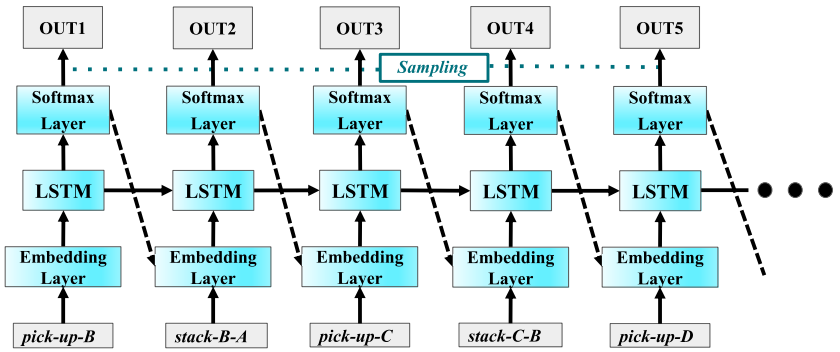

Fig. 4. The framework of our RNNPlanner approach.

is a hidden-output gate matrix, W_{co} is a cell-output gate matrix. b_i, b_f, b_c, and b_o are bias terms. Note that the matrices from cell to gate vectors (i.e., W_{ci}, W_{cf}, and W_{co}) are diagonal, such that each element e in each gate vector only receives input of element e of the cell vector. The major innovation of LSTM is its memory cell c_t, which essentially acts as an accumulator of the state information. c_t is accessed, written, and cleared by self-parameterized controlling gates, i.e., *input, forget, output* gates. Each time a new input x_t comes, its information is accumulated to the memory cell if the input gate i_t is activated. The past cell status c_{t-1} could be forgotten in this process if the forget gate f_t is activated. Whether the latest cell output c_t is propagated to the final state h_t is further controlled by the output gate o_t. The benefit of using the memory cell and gates to control information flow is that the gradient is trapped in the cell and prevented from vanishing too quickly.

5.2 Discovering Underlying Plans with the RNN Model

With the distributed representations of actions addressed in Section 3, we view each plan in the plan library as a sequence of actions, and the plan library as a set of action sequences that can be utilized to train the RNN model. The framework of RNN with sequences of actions can be seen from Figure 4, which is similar to Reference [25]. The bottom row in Figure 4 is an input action sequence "pick-up-B, stack-B-A, pick-up-C, stack-C-B, pick-up-D." The "Embedding Layer" computes vector representations of actions, which is pretrained by Section 3. Similar to a classic RNN cell, the LSTM cell feeds its output to both itself as a hidden state, and the softmax layer to obtain a probability distribution over all actions in the action vocabulary $\mathcal{A}$. From the perspective of the LSTM cell at the next step, it receives a hidden state from the previous step h_{t-1}, an action vector at the current step x_t. To obtain the index of most probable action, our model samples over the action distribution output from the softmax layer. That retrieved index could be mapped to an action in the vocabulary $\bar{\mathcal{A}}$.

The top row in Figure 4 is the output sequence, denoted by "OUT1, OUT2, OUT3, OUT4, OUT5, ...", which corresponds to the estimated sequence "$y_1, y_2, y_3, y_4, y_5, \ldots$" in Equation (17). Note that we exploit the dotted arrow to indicate two folds of meanings in Figure 4. When training the RNN model, the one pointed by the head of the dotted arrow (the embedding of the input) is used to compute the cross entropy with the output at the tail (output of LSTM cell at the previous step), and next-step observation as the input at the head to train the model. When using the trained RNN model to discover unknown actions, the model assumes what it predicts is the real next input and takes it to continue its prediction. Thus, the one pointed by the head is copied and identical to the one denoted by the tail. For example, the embedding of "stack-B-A" is copied from the prediction

vector of "OUT1" if the input "stack-B-A" was unknown. In addition, the arrows between each of two LSTM cells show the unrolling of an LSTM cell. The horizontal dashed line suggests that we obtain the action output at each step by sampling from probability distribution provided by the softmax layer. With the trained RNN model, we can discover underlying actions by simply exploiting the RNN model to generate unknown actions based on observed or already discovered actions.

An overview of our `RNNPlanner` algorithm can be seen from Algorithm 2. In Algorithm 2, Step 1 is the same as Step 1 of Algorithm 1. In Step 2, we let $y_t = x_{t+1}$, such that, given $p(x_{t+1}|y_t) = 1$, Equation (17) can be calculated by

$$\mathcal{L}(\mathbf{x};\theta) = -\sum_{t=1}^{T} log\text{LSTM}(x_{t+1}|x_{1:t};\theta), \tag{17}$$

where $\mathbf{x}$ is a plan in $\mathcal{L}$. We use MLE (Maximal Likelihood Estimation) to estimate parameters θ given $x_{1:t}$ as input and x_{t+1} as output for each $t \in [1, |\mathbf{x}|)$, where $\mathbf{x}$ is a plan from $\mathcal{L}$.

ALGORITHM 2: Framework of our `RNNPlanner` algorithm

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

1: learn vector representation of actions based on $\mathcal{L}$
2: initialize θ with random values
3: **while** the maximal number of repetitions Λ is not reached **do**
4: **for** each plan $p \in \mathcal{L}$ **do**
5: update parameters θ by optimizing the objective $\mathcal{L}(\mathbf{x};\theta)$ based on the vector representations of p
6: **end for**
7: **end while**
8: let $\tilde{p} = NULL$
9: **for** each $t = 1$ to $|O|$ **do**
10: **if** $o_t \in O$ is ϕ **then**
11: generate y_t based on $LSTM(y_t|\tilde{p};\theta)$
12: $o_t = y_t$
13: **end if**
14: $\tilde{p} = [\tilde{p}|o_t]$
15: **end for**
16: **return** $\tilde{p}$

For example, given the observation,

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

we can generate the first ϕ based on *pick-up-B*, the second ϕ based on actions from *pick-up-B* to *put-down-D*, the third ϕ based on all previous actions (including the generated actions for ϕs).

The running time complexity of Step 1 in Algorithm 2 is the same as addressed in Algorithm 1, i.e., $O(Iters \times T_{max} \times |\mathcal{L}| \times WinSize)$. The time cost of Steps 2 to 7 of Algorithm 2 is $O(\Lambda \times |\mathcal{L}| \times T_{max}^2)$, where T_{max}^2 is the time cost of updating θ (Step 5) based on plan p, since it needs to scan each action a in p and a's preceding actions when computing the objective $\mathcal{L}(\mathbf{x};\theta)$ as shown in Equation (17). The time cost of Steps 8 to 15 is $O(|O|^2)$ based on the parameters θ learnt by previous steps. In summary, the complexity of Algorithm 2 is $O(\Lambda \times |\mathcal{L}| \times T_{max}^2)$, considering T_{max} is much larger than $WinSize$, and the length of O is in general much less than $|\mathcal{L}| \times T_{max}^2$.

6 EXPERIMENTS

In this section, we evaluate our DUP and RNNPlanner algorithms in three planning domains from International Planning Competition, i.e., blocks,[1] depots,[2] and driverlog,[2] which are described as follows [74]:

- Blocks: The objects in the domain include a finite number of cubical blocks and a table large enough to hold all of them. Each block is on a single other object (either another block or the table). For each block, either it is clear or else there is a unique block a sitting on it. There is one kind of action: move a single clear block, either from another block onto the table or from an object onto another clear block. As a result of moving block b from c onto d, b is sitting on d instead of c, c is clear (unless it is the table), and d is not clear (unless it is the table) [26].
- Depots: The domain consists of actions to load and unload trucks, using hoists that are available at fixed locations. The loads are all crates that can be stacked and unstacked onto a fixed set of pallets at the locations. The trucks do not hold crates in a particular order, so they can act like a table in the Blocks domain, allowing crates to be reordered.
- Driverlog: This domain has drivers that can walk between locations and trucks that can drive between locations. Walking requires traversal of different paths from those used for driving, and there is always one intermediate location on a footpath between two road junctions. The trucks can be loaded or unloaded with packages (with or without a driver present), and the objective is to transport packages between locations, ending up with a subset of the packages, the trucks, and the drivers at specified destinations.

To generate training and testing data, we randomly created 5K planning problems for each domain and solved these planning problems with a planning solver (we used FF[3] in the experiment), to produce 5K plans.

We define the accuracy of our DUP and RNNPlanner algorithms as follows: For each unobserved action $\bar{a}_x$, DUP and RNNPlanner suggest a set of possible actions S_x that 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., $a_{truth} \in S_x$, then we increase the number of correct suggestions 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 sizes of S_x in the experiment. Note that we borrow the name of "accuracy" with a different definition from the data-mining community, where accuracy is defined by the ratio of *True Positive* and *True Negative* over all testing data. In our definition of *accuracy*, we view all actions in the recommended set as equally important and do not take their corresponding values into account once they are recommended.

We randomly divided the plans into 10 folds, with 500 plans in each fold. We ran our DUP algorithm 10 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., O) with a specific percentage ξ of the plan length. Features of datasets are shown in Table 1, where the second column, denoted by "#plan," is the number of plans generated; the third column, denoted by "#word," is the total number of words (or actions) of all plans; the fourth column, denoted

[2] http://ipc02.icaps-conference.org/CompoDomains/IPC3.tgz.
[3] https://fai.cs.uni-saarland.de/hoffmann/ff.html.

Table 1. Features of the Three Domains: Blocks, Depots, and Driverlog

domain	#plan	#word	avg-of-length	#vocabulary
blocks	5,000	292,250	58	1,250
depots	5,000	209,711	42	2,273
driverlog	5,000	179,621	36	1,441

by "avg-of-length," is an average of length over all of the plans; and the last column, denoted by "#vocabulary," is the size of vocabulary used in all plans. In the experiment, we evaluated our approaches on their performances with respect to the three datasets whose vocabulary sizes are different from each other.

We set the learning constant δ to be 0.1 and the maximal number of iterations to be 1500 in DUP. In RNNPlanner, we set the size of the LSTM hidden layer to be 800, training epoch number to be 250, learning rate to be 0.002, decay rate to be 0.97, and batch size to be 50. Detailed settings of DUP[4] and RNNPlanner[5] can be found online, where the source codes and datasets are available. We compared our approaches to baselines as shown below:

- MatchPlan: 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, 23], which we denote by MatchPlan. We develop a MatchPlan system based on the high-level idea shared by References [9, 23], aiming at recognizing plans from "flat" plan libraries (i.e., the plans in the libraries are total-order action sequences), which DUP is designed for, and compare our DUP algorithm to MatchPlan with respect to different percentages of unobserved actions ξ and different sizes of suggestion or recommendation set S_x.
- ARMS+PRP: Another baseline is action-models–based plan recognition approach [55] (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 [65] to learn action models from the plan library and fed the action models to the PRP approach. We call this hybrid plan recognition approach ARMS+PRP. To learn action models, ARMS requires 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. To the best of our knowledge, there is no off-the-shelf approach doing this. We thus built this hybrid system based on off-the-shelf action model learning approach ARMS and plan-recognition-as-planning approach PRP to be compared with. We believe replacing PRP with other approaches will not change the accuracy, since the accuracy depends on the learnt "complex" (incomplete) action models that influence the plan recognition quality essentially. We set to be 0.5 the probability threshold of the frequent set-mining algorithm in ARMS when using ARMS to learn action models, which has been demonstrated effective by Reference [65]. 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] http://xplan-lab.org//resource/shallow-plan-release.zip.
[5] https://github.com/YantianZha/Discovering-Underlying-Plans-Based-on-Shallow-Models.

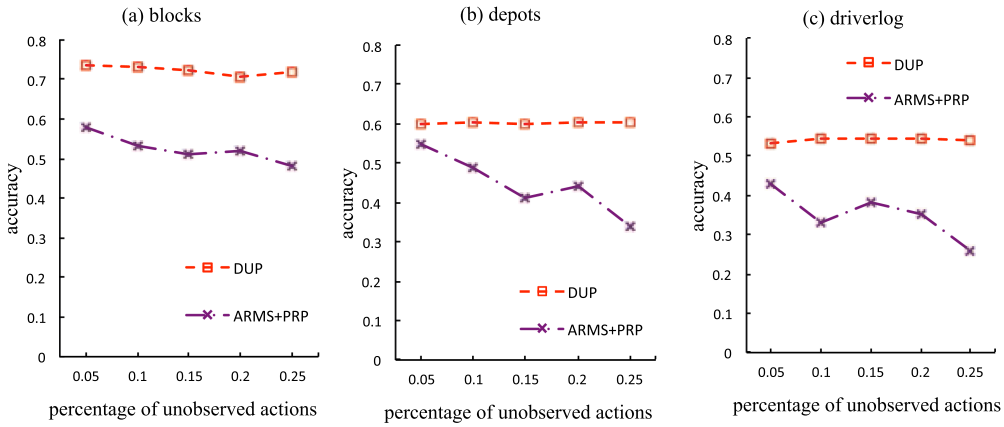

Fig. 5. Accuracies of DUP and ARMS+PRP with respect to different percentage of unobserved actions.

In the following subsections, we aim to evaluate our approaches DUP and RNNPlanner in the following aspects:

- We first compared DUP to ARMS+PRP to illustrate that the shallow model learnt by DUP can indeed outperform the "exact" model learnt by ARMS+PRP.
- We then compared DUP to MatchPlan to show that the shallow model learnt by DUP performs better than directly searching methods as done by MatchPlan.
- After that, we compared DUP and RNNPlanner to analyze the advantages and disadvantages of the two shallow-model learning approaches.
- Finally, we compared our proposed DUP and RNNPlanner approaches to SPR, a state-of-the-art approach, to exhibit the effectiveness of our approaches on generalizing to recognize plans that are not from the plan library.

6.1 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.

6.1.1 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 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 10, and the learning constant δ in Equation (7) to be 0.1. For ARMS+PRP, we generated 20 candidate goals for each testing example, including the ground-truth goal that corresponds to the ground-truth plan to be recognized. The results are shown in Figure 5.

From Figure 5, we can see that in all three domains, the accuracy 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 that are often with noise. When feeding those noisy models to PRP,

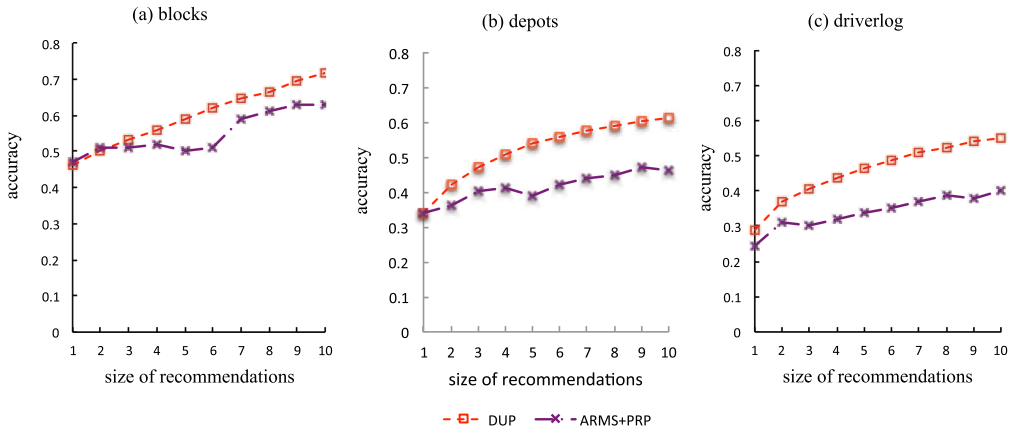

Fig. 6. Accuracies of DUP and ARMS+PRP with respect to different size of recommendations.

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 5 (as well as Figure 7), 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 5 (likewise for Figure 7).

6.1.2 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 the 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 (7) 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. The results are shown in Figure 6.

From Figure 6, 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

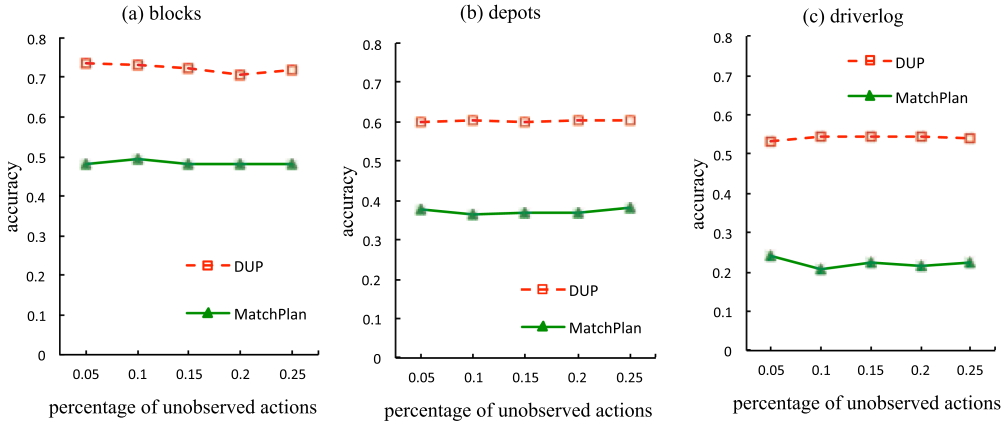

Fig. 7. Accuracies of DUP and MatchPlan with respect to different percentage of unobserved actions.

intuition, since the larger the recommended set is, the higher the possibility for the *true* action to be in the recommended set. We can also see that the accuracy of our DUP algorithm is 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 5 in the previous section; that is, the "shallow" model learnt by our DUP algorithm is better for recognizing plans than both the "exact" planning models learnt by ARMS for recognizing plans with planning techniques. Furthermore, the advantage of DUP becomes even larger when the size of the recommended action set increases, which suggests our vector representation–based learning approach can better capture action relations when the size of the 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.

6.2 Comparison between DUP and MatchPlan

In this experiment, we compare DUP to MatchPlan, which is built based on the idea of Reference [23]. 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.

6.2.1 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 10, and the learning constant δ in Equation (7) 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 O, MatchPlan matches the previous three actions and subsequent three actions of $\bar{a}_x$ to plans in $\mathcal{L}$ and recommends 10 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 7.

From Figure 7, 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

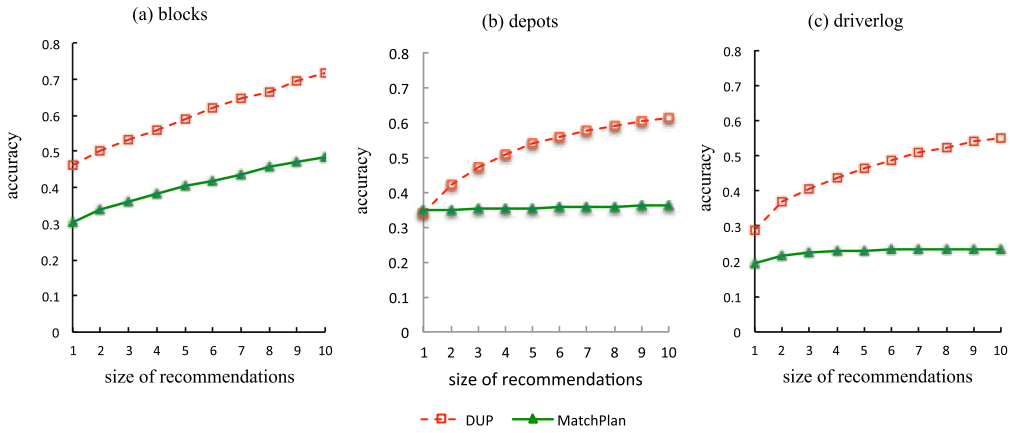

Fig. 8. Accuracies of DUP and MatchPlan with respect to different size of recommendations.

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 5), 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.

6.2.2 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 the 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 8.

We can observe that the accuracy of our DUP algorithm is generally higher than MatchPlan in all three domains in Figure 8, 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 increases. In other words, larger size of recommendations provides better chance for "shallow" models learnt by DUP to perform better.

6.2.3 Varying Size of Training Set. To see the effect of size of training set, we ran both DUP and MatchPlan with different sizes of training sets. We used the same setting as done by the last subsection except fixing the size of recommendations to be 10 when running both algorithms. We varied the size of the training set from 2500 to 4500. The results are shown in Figure 9.

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

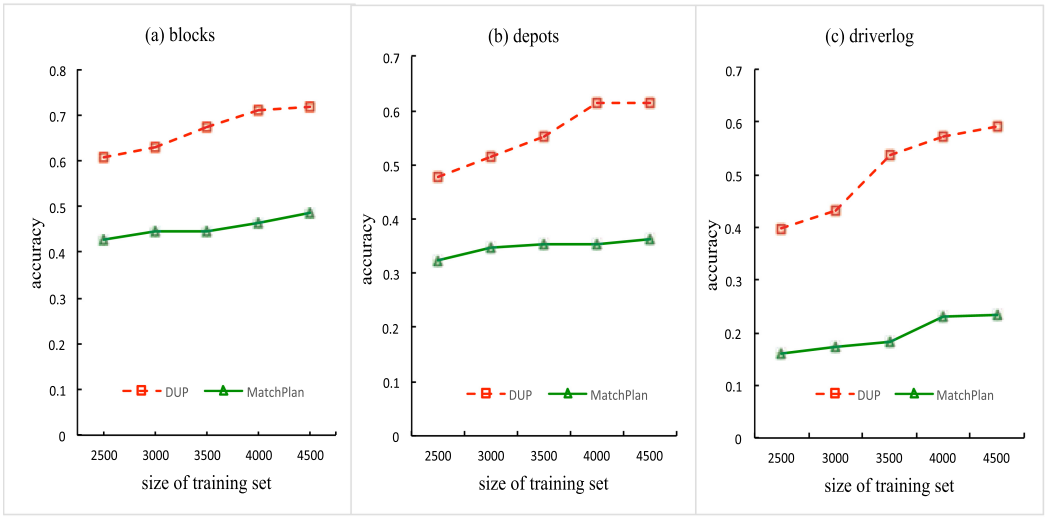

Fig. 9. Accuracies of DUP and MatchPlan with respect to different size of training set.

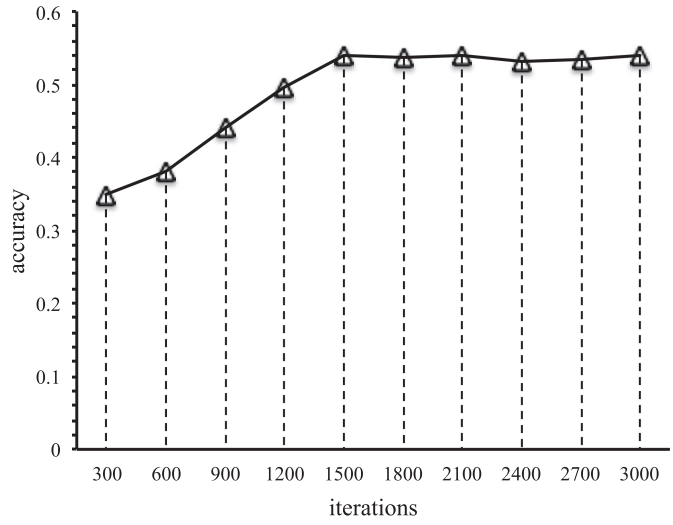

Fig. 10. Accuracy with respect to different number of iterations in the blocks domain.

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

6.2.4 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 results in the *blocks* domain (the results of the other two domains are similar) in Figure 10.

From Figure 10, 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.

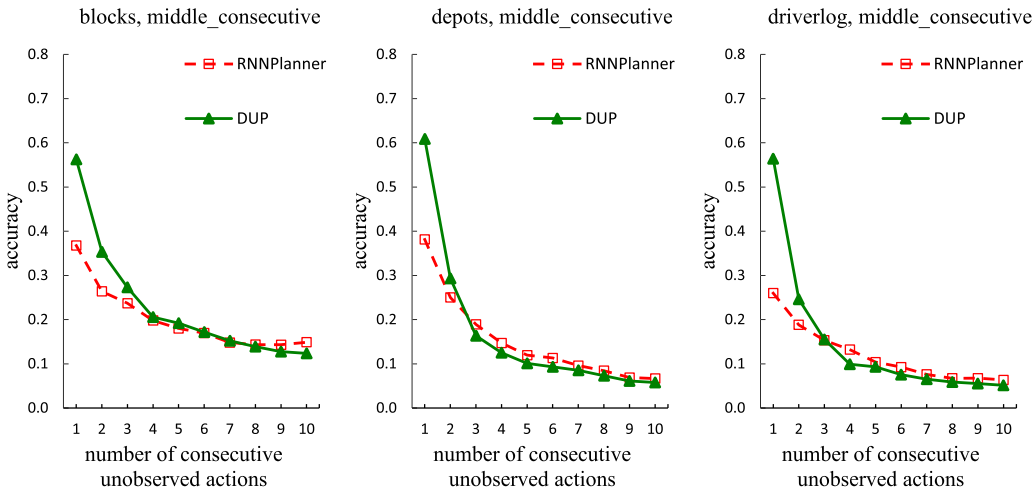

Fig. 11. Accuracy with respect to missing actions in the middle.

6.3 Comparison between RNNPlanner and DUP

In this section, we compare RNNPlanner with DUP to see the change of performance with respect to different distributions of missing actions in the underlying plans to be discovered. In this experiment, we are interested in evaluating the performance on consecutive missing actions in the underlying plans, since these scenarios often exist in many applications such as surveillance [1]. We first test the performance of both RNNPlanner and DUP in discovering underlying plans with only consecutive missing actions in the "middle" of the plans, i.e., actions are not missing at the end or in the front, which indicates missing actions can be inferred from both previously and subsequently observed actions. Then, we evaluate both RNNPlanner and DUP in discovering underlying plans with only consecutive missing actions at the end of the plans, which indicates missing actions can only be inferred from previously observed actions. After that, we also evaluate the performances of our RNNPlanner and DUP approaches with respect to the size of the recommendation set. In the following subsections, we present the experimental results regarding those three aspects.

6.3.1 Performance with Missing Actions in the Middle. To see the performance of RNNPlanner and DUP in cases when actions are missing in the middle of the underlying plan to be discovered, we vary the number of consecutive missing actions from 1 to 10 to see the change of accuracies of both RNNPlanner and DUP. We set the window size to be 1 and the recommendation size to be 10. The results are shown in Figure 11.

From Figure 11, we can see that the accuracies of both RNNPlanner and DUP generally become lower when the number of consecutive unobserved actions increases. This is consistent with our intuition, since the more actions are missing, the less information can be used to help infer the unobserved actions, which results in low accuracies. Comparing the curves of RNNPlanner and DUP, we can see that the accuracy of DUP is higher than RNNPlanner at the beginning; this is because DUP exploits information of both observed actions before and after missing actions to infer the missing actions, while RNNPlanner just exploits observed actions before missing actions. When the number of missing actions is larger than three, the accuracy of DUP is lower than RNNPlanner; this is because the window size of DUP is set to be 1, which indicates we exploit one action before the missing actions and one action after the missing actions to estimate the missing actions. When

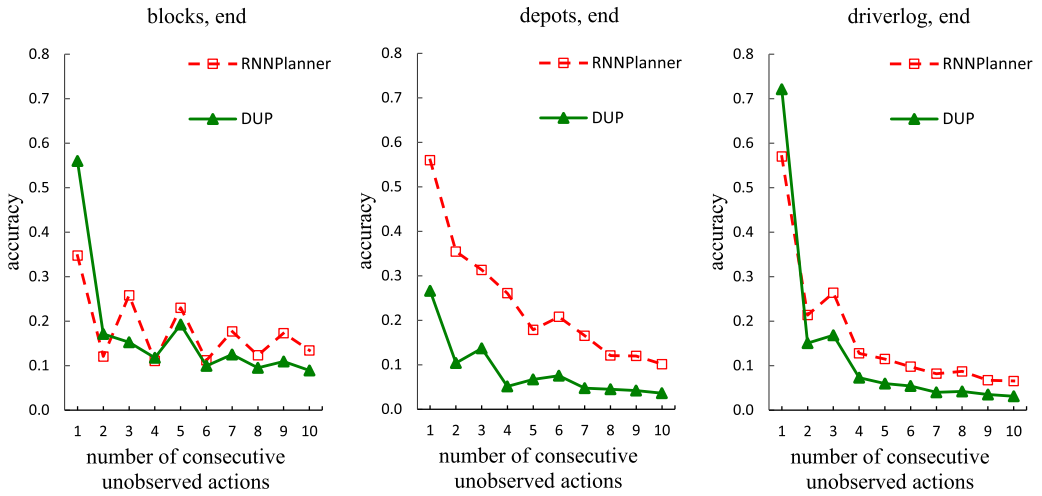

Fig. 12. Accuracy with respect to missing actions in the end.

the consecutive missing actions are more than 1, there may not be sufficient context information for inferring the missing actions, resulting in low accuracies.

6.3.2 Performance with Missing Actions at the End. We also would like to see the performance of RNNPlanner and DUP in discovering missing actions at the end, which is prevalent in application domains that aim at discovering/predicting future actions. Similar to previous experiments, we vary the number of consecutive unobserved or missing actions to see the change of accuracies of both RNNPlanner and DUP. We set the window size to be 1 and the recommendation size to be 10 as well. The experimental results are shown in Figure 12.

From Figure 12, we can observe that the accuracies of both RNNPlanner and DUP generally decrease when the number of consecutive missing actions increases. This is similar to previous experimental results; that is, the more actions are missing, the less information is available for estimating the missing actions, which results in lower accuracy. In addition, we can also see that RNNPlanner generally performs better than DUP, which indicates that the RNNs-based approach, i.e., RNNPlanner, can indeed better exploit observed actions to predict future missing actions, since RNNs are capable of flexibly leveraging long- or short-term information to help predict missing actions. We can also see that the performance of both RNNPlanner and DUP decreases sharply when the number of consecutive missing actions is larger than three; this is because we set the window size to be 1 (i.e., we consider three consecutive actions each time we calculate the *a posteriori* probability), which indicates we exploit one action before the missing actions and one action after the missing actions to estimate the missing actions. When the consecutive missing actions are more than 1, there may not be sufficient context information for inferring the missing actions, resulting in low accuracies. Based on our experimental results of ARMS+PRP and MatchPlan (cf. Figures 5 and 7), they are even worse than our RNNPlanner and DUP when the window size is set to be 1.

6.3.3 Performance with Respect to Different Recommendation Size. To see the change with respect to different recommendation size, we vary the size of recommendation sets from 1 to 10 and calculate their corresponding accuracies. We test our approaches with four cases: A. there are five actions missing at the end; B. there are five actions missing in the middle; C. there is one action missing at the end; D. there is one action missing in the middle. The results are shown in Figures 13–16 corresponding to cases A–D, respectively.

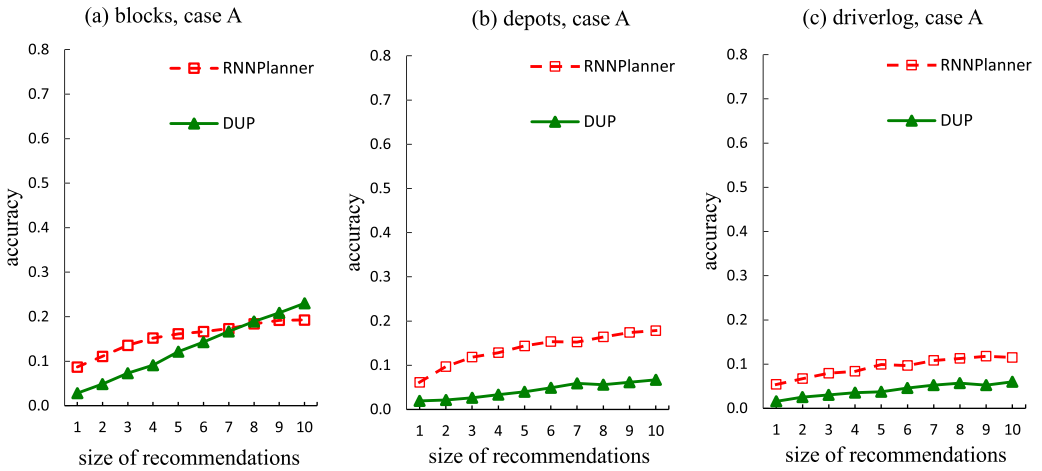

Fig. 13. Case A: accuracy with respect to different size of recommendations.

Case A: As shown in Figure 13, RNNPlanner performs better than DUP mostly, except for when the recommendation set size is larger/equal to eight in blocks domain; this is because RNNPlanner, which contains LSTM cells, is able to actively remember or forget past observations (inputs) and computations (hidden states). For example, if in a set of sequences, a pattern A^{**} follows A^{*} after three words (i.e., $\ldots, A^{*}, w_i, w_{i+1}, w_{i+2}, A^{**}, \ldots$), where w_i, w_{i+1}, w_{i+2} could be any word from the vocabulary except for A^{*} and A^{**}. And if the window size of DUP is smaller than three, then DUP is not able to utilize this pattern to predict A^{**} mainly based on A^{*}. When predicting the A^{**}, the DUP with context size one, works by searching for the most similar word to the w_{i+2}. One would yet argue that we can set a larger window size for DUP. Larger window size does not necessarily lead to higher accuracy, since using a larger window size also adds more noise in the training of DUP. Remember that the word2vec model treats equally all possible word pair samples within its context window.

In addition, observing the accuracies (in terms of the size of recommendation S_x) of all three domains, we can see only in the *blocks* domain that DUP outperforms RNNPlanner when S_x is larger than eight. Also in the *blocks* domain, DUP has the best performance, comparing to how DUP functions in the other two domains; this is because plans from the *blocks* domain has an overall higher ratio of #word to #vocabulary, which increases the possibility that the word pattern outside a context window would reappear inside the window and consequently help DUP recognize actions in the missing positions. Coming back to the example when we have a plan like $\ldots, A^{*}, w_i, w_{i+1}, w_{i+2}, A^{**}, \ldots$, in *blocks* domain, it is more possible the word A^{*} happens again in one of w_i, w_{i+1}, and w_{i+2}.

Case B: What we can observe here from Figure 14 is similar to our observations in ***case A***. RNNPlanner generally performs better than DUP, except for when the size of recommendation S_x is larger or equal to nine in *blocks* domain. It could also be observed that both RNNPlanner and DUP have the best accuracy performance in the *blocks* domain.

And by comparing Figure 14 in ***case B*** (five removed actions in the middle) with Figure 13 in ***case A*** (five removed actions at the end), we can see that the accuracy difference between DUP and RNNPlanner at each size of recommendation along the x-axis is smaller in ***case B***; this is because RNNPlanner only leverages the observed actions before a missing position, whereas DUP has the advantage of additionally using the observation after a missing position.

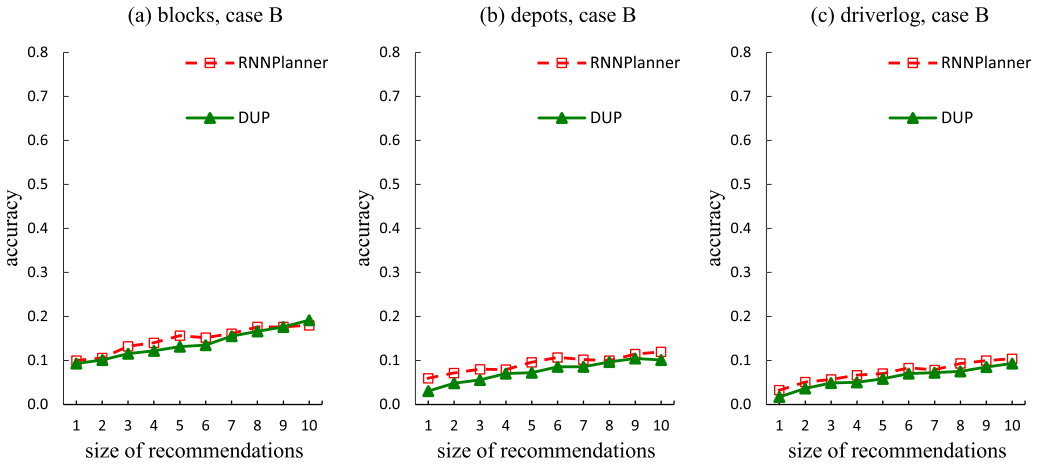

Fig. 14. Case B: accuracy with respect to different size of recommendations.

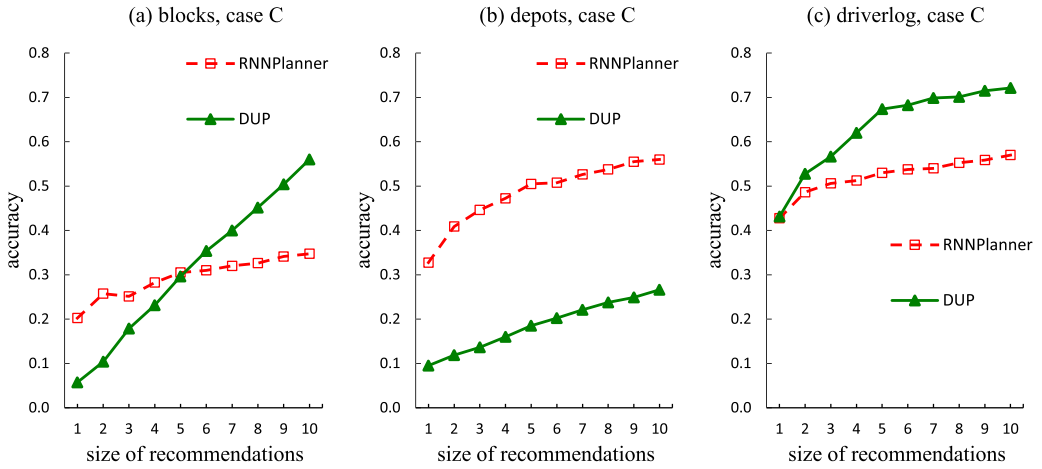

Fig. 15. Case C: accuracy with respect to different size of recommendations.

Case C: From Figure 15, we can see that both RNNPlanner and DUP could outperform each other in certain domains and recommendation set sizes (S_x). In *blocks* domain, DUP is better when S_x is larger than five. In *depots* domain, RNNPlanner is overwhelmingly better than DUP. In *driverlog* domain, DUP performs better overall except that, when there is only one recommendation, DUP is as good as RNNPlanner; this is because the domains *blocks* and *driverlog* are not as complex as *depots*, and DUP performs better in relative simple domains while worse in complex domains compared to RNNPlanner.

Case D: From the results in Figure 16, we can see that DUP functions better than RNNPlanner over all three domains, whereas DUP is worse in ***case A*** and ***case B***, and could occasionally be better than RNNPlanner in ***case C***. It makes sense in that, on the one hand, within the fixed and short context window, if there are much less positions with removed actions, DUP would have an improved performance. On the other hand, RNNPlanner is not able to leverage the information from both sides of a position with a missing action. Therefore, in ***case D***, DUP gains the benefit

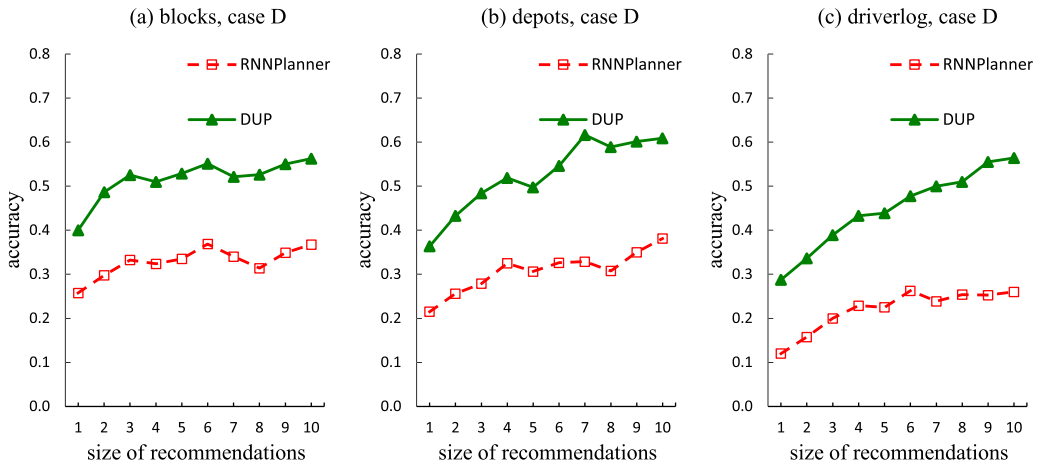

Fig. 16. Case D: accuracy with respect to different size of recommendations.

from both assumptions that there is only one missing action, and the position of that action is randomly chosen in the middle of a plan.

7 RELATED WORK

Our work is related to the four aspects on plan recognition, i.e., plan recognition with plan library, plan recognition with domain knowledge, sensor-based activity/plan recognition, and planning with incomplete domain models. We will introduce the four aspects in the following subsections, respectively.

7.1 Plan Recognition with Plan Library

Our work is related to plan recognition with plan library. 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" [34]. All actions and 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 work computationally [37]. They automatically constructed plan-library from domain primitives, which was different from Reference [34], 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 [30] proposed an anytime plan recognition algorithm to reduce the number of generated plan execution models based on weighted model counting. Avrahami-Zilberbrand and Kaminka [9] considered the observed agent to be an adversary and built an efficient hybrid adversarial plan recognition system that is composed of two processes, i.e., a plan recognizer capable of efficiently detecting anomalous behavior, and a utility-based plan recognizer incorporating the observer's own biases. Mirsky and Gal [44] proposed an efficient algorithm for online plan recognition called Semi-Lazy Inference Mechanism, which combined both bottom-up and top-down parsing processes. Massardi et al. propose an anytime top-down approach [42] to deal with noisy observations based on the plan library. These approaches are, however, difficult to represent uncertainty and recognize plans that are not from the plan library. Although we exploit a library of plans in our DUP and RNNPlanner approaches, we aim to learn 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.

7.2 Plan Recognition with Domain Knowledge

Instead of using a library of plans, Ramirez and Geffner [55] 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 [65, 79]). Due to the computational cost of calling a planner twice for each possible goal in Reference [55], E-Martín et al. [20] proposed an approach that can quickly provide a probability distribution over the possible goals by computing cost estimates using a plan graph. Considering unreliability of a sensor malfunction or intentional obfuscation by malware, Sohrabi et al. [61] proposed a relaxation of the plan-recognition-as-planning formulation that allows unreliable observations. Pereira et al. [50] proposed two goal-recognition heuristics based on planning techniques that rely on planning landmarks [28]. Saria and Mahadevan presented hierarchical multi-agent Markov processes as a framework for hierarchical probabilistic plan recognition in cooperative multi-agent systems [57]. Singla and Mooney proposed an approach to abductive reasoning using a first-order probabilistic logic to recognize plans [59]. Amir and Gal addressed a plan recognition approach to recognizing student behaviors using virtual science laboratories [7]. Ramirez and Geffner exploited off-the-shelf classical planners to recognize probabilistic plans [56]. Albrecht and Stone recently provided a survey on plan recognition in three aspects, i.e., plan recognition in hierarchical plan libraries, plan recognition by planning in action models, and plan recognition by similarity to past plans [3]. Höller et al. propose to combine the expressive and compact grammar-like HTN representation with the advantage of planning techniques for plan and goal recognition [29]. To alleviate the requirement of complete domain models, Pereira et al. propose to recognize goals using incomplete domain theories by considering different notions of planning landmarks [51]. Different from those approaches, we do not require any domain model knowledge provided as input. Instead, we automatically learn shallow action models from previous plan cases for recognizing unknown plans that may not be identical to previous cases.

7.3 Sensor-based Activity Recognition

Activity recognition with sensors has been a major focus in the area of artificial intelligence. There has been an increasing interest in inferring a user's activities through low-level sensor modeling. Liao et al. applied a dynamic Bayesian network to estimate a person's locations and transportation modes [40] from logs of GPS data with relatively precise location information. Bui et al. introduced an abstract hidden Markov model to infer a person's goal from camera data in an indoor environment, but it is not clear from the article how action sequences are obtained from camera data [12]. Yin et al. explicitly relied on training a location-based sensor model [66] to infer locations from signals; the locations are part of the input that can serve as labels in the training data. To reduce the human labeling effort and cope with the changing signal profiles when the environment changes, Yin et al. dealt with the second issue by transferring the labeled knowledge [67] between time periods. Pan et al. proposed to perform location estimation [47, 48] through online co-localization and applied multi-view learning for migrating the labeled data to a new time period.

Different from considering location information, Lester et al. propose to build user models [38] for different users and recognize user activities based on the models. They treat all of the users equally by simply mixing their data in training. However, different users may behave differently given similar sensor observations. For example, a user may visit a coffee shop for a meal and another user just enjoys sitting in its outdoor couches to read a research paper. These two users probably observe similar Wi-Fi signals, but their activities are quite personalized. This implies that

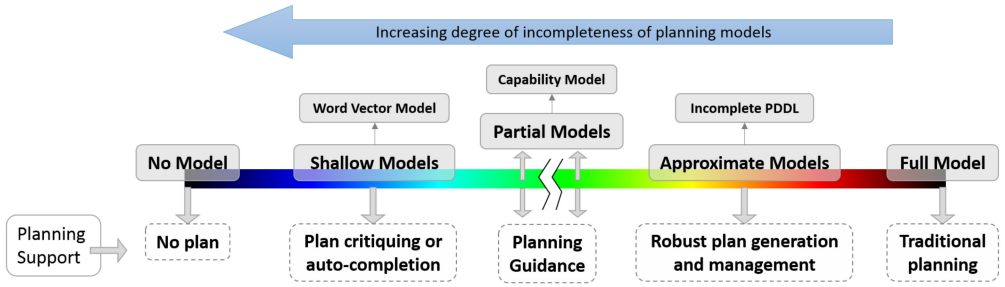

Fig. 17. Schematic view of incomplete models and their relationships in the spectrum of incompleteness.

it may not be appropriate to require all of the users to share one common, user-independent activity recognizer. Therefore, Zheng et al. proposed to build a personalized activity recognition model [70] by considering the relations among users. Using sensor data as input, Hodges and Pollack designed machine learning–based systems [27] for identifying individuals as they perform routine daily activities such as making coffee. Liao et al. proposed to infer user transportation modes [40] from readings of Radio-Frequency Identifiers (RFIDs) and Global Positioning Systems (GPSs). Freedman et al. explore the application of Natural Language Processing (NLP) techniques [21], i.e., Latent Dirichlet Allocation topic models, to human skeletal data of plan execution traces obtained from an RGB-D sensor. Bulling et al. discussed the key research challenges [13] that human activity recognition shared with general pattern recognition. When activity recognition is performed indoors and in cities using the widely available Wi-Fi signals, there is much noise and uncertainty. Xie et al. proposed a temporal-then-spatial recalibration scheme to build end-to-end Memory Attention Networks (MANs) [64] for solving skeleton-based action recognition tasks. Chen et al. propose a multi-agent spatial-temporal attention model [14] to jointly recognize activities of multiple agents. To tackle the limitations of feature extraction and training data labeling effort, Amado et al. [5, 6] propose to combine goal recognition techniques and deep auto-encoders to explore unsupervised learning to generate domain theories from data and use the resulting domain theories to deal with incomplete and noisy observations. Qian et al. propose a distribution-based semi-supervised learning approach [54] to recognize human activities.

Many different applications of activity recognition have been studied by researchers. For example, Pollack et al. show that home-based rehabilitation [53] can be provided for people suffering from traumatic brain injuries by automatically monitoring human activities. Chu et al. present a model of interactive activity recognition [15] to determine the user's state by interpreting sensor data and/or by explicitly querying the user. The system can be used in an assistive system for persons with cognitive disabilities, which can prompt the user to begin, resume, or end tasks. Zheng et al. proposed to recognize physical activity from accelerometer data [71] using a Multi-Scale Ensemble Method.

Different from the above-mentioned sensor-based activity/plan recognition, we do not assume we have any labeled sensing data for recognizing plans. Instead, we focus on symbolic plan recognition in this article.

7.4 Planning with Incomplete Action Models

Our work is also related to planning with incomplete action models (or model-lite planning [32, 77]). Figure 17 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 [63]. 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 [68]. 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 article 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 (cf. Reference [19]) 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 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.

8 CONCLUSION AND DISCUSSION

In this article, we present two novel plan recognition approaches, `DUP` and `RNNPlanner`, based on vector representation of actions. For `DUP`, 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. For `RNNPlanner`, we let the neural network itself learn the word embedding, which would then be utilized by higher LSTM layers. We also empirically exhibit the effectiveness of our approaches.

In the future, it would be interesting to consider future studies as shown below:

- While we focused on a one-shot recognition task in this article, in practice, human-in-the-loop planning will consist of multiple iterations, with `DUP` and `RNNPlanner` 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` and `RNNPlanner` recognition algorithms "incremental."
- The word-vector–based action model we developed in this article provides interesting contrasts to the standard precondition- and effect-based action models used in the automated planning community. One of our future aims is to provide a more systematic comparison of the tradeoffs offered by these models. In this paper we focused on "plan recognition" aspects of this model, assuming "planning support" is limited to suggesting potential actions to the humans. In the future, we will also consider "critiquing" the plans being generated by 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., Reference [52]), and will have to develop justifications analogous to those used in recommendation systems.
- Our current study shows that even direct application of word vector learning methods provides 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.

- Another potential application for the type of distributed action representations proposed in this article is social media analysis. In particular, work such as Reference [35] 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 article provide a promising way of handling this challenge.
- In this article, we considered distributed representations of actions. In many real-world applications, however, information in the form of images or texts describing "states" between actions is ubiquitous. It would be interesting to explore representations of states—via deep learning models [36], for example—to help recognize plans together with distributed representations of actions.
- The plan to be recognized was assumed to be executed by a single agent in this article. We believe that the distributed representations of actions could be extended into handling team plan recognition [39, 73].
- In our `RNNPlanner`, we exploit a specific RNN model, LSTM, to estimate unobserved actions based on distributed representations of actions. Considering the effectiveness of other specific RNN-based models, such as GRU [16], it would be interesting to study the feasibility of exploring other RNN-based models to improve the recognition accuracy.
- In our `DUP` and `RNNPlanner`, the models are black boxes to humans. It would be interesting to study the explicability of the models [69] and consider human awareness [72] in the future while recognizing underlying plans.

REFERENCES

[1] Besma R. Abidi, Nash R. Aragam, Yi Yao, and Mongi A. Abidi. 2009. Survey and analysis of multimodal sensor planning and integration for wide area surveillance. *ACM Comput. Surv.* 41, 1, Article 7 (Jan. 2009), 36 pages.

[2] Stefano V. Albrecht and Subramanian Ramamoorthy. 2015. Are you doing what I think you are doing? Criticising uncertain agent models. In *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence (UAI'15)*. 52–61.

[3] Stefano V. Albrecht and Peter Stone. 2018. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artif. Intell.* 258 (2018), 66–95. `DOI`: https://doi.org/10.1016/j.artint.2018.01.002

[4] Rabah Alzaidy, Cornelia Caragea, and C. Lee Giles. 2019. Bi-LSTM-CRF sequence labeling for keyphrase extraction from scholarly documents. In *Proceedings of the World Wide Web Conference (WWW'19)*. 2551–2557.

[5] Leonardo Amado, João Paulo Aires, Ramon Fraga Pereira, Mauricio Cecilio Magnaguagno, Roger Granada, and Felipe Meneguzzi. 2018. LSTM-based goal recognition in latent space. *CoRR* abs/1808.05249 (2018). arxiv:1808.05249 Retrieved from http://arxiv.org/abs/1808.05249.

[6] Leonardo Amado, Ramon Fraga Pereira, João Paulo Aires, Mauricio Cecilio Magnaguagno, Roger Granada, and Felipe Meneguzzi. 2018. Goal recognition in latent space. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'18)*. 1–8. `DOI`: https://doi.org/10.1109/IJCNN.2018.8489653

[7] Ofra Amir and Yaakov (Kobi) Gal. 2011. Plan recognition in virtual laboratories. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'11)*. 2392–2397.

[8] Masataro Asai and Alex Fukunaga. 2018. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI'18), the 30th Innovative Applications of Artificial Intelligence (IAAI'18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI'18)*. 6094–6101. Retrieved from https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16302.

[9] Dorit Avrahami-Zilberbrand and Gal A. Kaminka. 2014. Keyhole adversarial plan recognition for recognition of suspicious and anomalous behavior.

[10] S. R. K. Branavan, Nate Kushman, Tao Lei, and Regina Barzilay. 2012. Learning high-level planning from text. In *Proceedings of the Meeting of the Association for Computational Linguistics (ACL'12)*.

[11] Hung H. Bui. 2003. A general model for online probabilistic plan recognition. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'03)*. 1309–1318.
[12] Hung Hai Bui, Svetha Venkatesh, and Geoff A. W. West. 2002. Policy recognition in the abstract hidden Markov model. *J. Artif. Intell. Res.* 17 (2002), 451–499.
[13] Andreas Bulling, Ulf Blanke, and Bernt Schiele. 2014. A tutorial on human activity recognition using body-worn inertial sensors. *ACM Comput. Surv.* 46, 3 (2014), 33.
[14] Kaixuan Chen, Lina Yao, Dalin Zhang, Bin Guo, and Zhiwen Yu. 2019. Multi-agent attentional activity recognition. *CoRR* abs/1905.08948 (2019). arxiv:1905.08948 Retrieved from http://arxiv.org/abs/1905.08948.
[15] Yi Chu, Young Chol Song, Richard Levinson, and Henry A. Kautz. 2012. Interactive activity recognition and prompting to assist people with cognitive disabilities. *J. Amb. Intell. Smart Envir.* 4, 5 (2012), 443–459.
[16] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR* abs/1412.3555 (2014). arxiv:1412.3555 Retrieved from http://arxiv.org/abs/1412.3555.
[17] Philip R. Cohen, Edward C. Kaiser, M. Cecelia Buchanan, Scott Lind, Michael J. Corrigan, and R. Matthews Wesson. 2015. Sketch-Thru-Plan: A multimodal interface for command and control. *Commun. ACM* 58, 4 (2015), 56–65.
[18] Xin Dong, Alon Y. Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. 2004. Similarity search for web services. In *Proceedings of the 30th International Conference on Very Large Data Bases*. 372–383.
[19] Xin Dong, Alon Y. Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. 2004. Similarity search for web services. In *Proceedings of the Very Large Data Bases Conference (VLDB'04)*. 372–383.
[20] Yolanda E-Martín, María D. R.-Moreno, and David E. Smith. 2015. A fast goal recognition technique based on interaction estimates. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*. 761–768. Retrieved from http://ijcai.org/Abstract/15/113.
[21] Richard G. Freedman, Hee-Tae Jung, and Shlomo Zilberstein. 2014. Plan and activity recognition from a topic modeling perspective. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS'14)*.
[22] Christopher W. Geib and Robert P. Goldman. 2009. A probabilistic plan recognition algorithm based on plan tree grammars. *Artif. Intell.* 173, 11 (2009), 1101–1132.
[23] Christopher W. Geib and Mark Steedman. 2007. On natural language processing and plan recognition. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*. 1612–1617.
[24] Alex Graves. 2013. Generating sequences with recurrent neural networks. *CoRR* abs/1308.0850 (2013). arxiv:1308.0850 Retrieved from http://arxiv.org/abs/1308.0850.
[25] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. 2013. Speech recognition with deep recurrent neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP'13)*. 6645–6649. DOI: https://doi.org/10.1109/ICASSP.2013.6638947
[26] Naresh Gupta and Dana S. Nau. 1992. On the complexity of blocks-world planning. *Artif. Intell.* 56, 2-3 (1992), 223–254. DOI: https://doi.org/10.1016/0004-3702(92)90028-V
[27] Mark R. Hodges and Martha E. Pollack. 2007. An "object-use fingerprint": The use of electronic sensors for human identification. In *Proceedings of the Conference on Ubiquitous Computing (UbiComp'07)*. 289–303.
[28] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. 2004. Ordered landmarks in planning. *J. Artif. Intell. Res.* 22 (2004), 215–278. DOI: https://doi.org/10.1613/jair.1492
[29] Daniel Höller, Gregor Behnke, Pascal Bercher, and Susanne Biundo. 2018. Plan and goal recognition as HTN planning. In *Proceedings of the IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI'18)*. 466–473. DOI: https://doi.org/10.1109/ICTAI.2018.00078
[30] Froduald Kabanza, Julien Filion, Abder Rezak Benaskeur, and Hengameh Irandoust. 2013. Controlling the hypothesis space in probabilistic plan recognition. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'13)*.
[31] Subbarao Kambhampati. 2007. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain models. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*. 1601–1605.
[32] Subbarao Kambhampati. 2007. Model-lite planning for the web age masses: The challenges of planning with incomplete and evolving domain theories. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'07)*. 1601–1605.
[33] Subbarao Kambhampati and Kartik Talamadupula. 2015. Human-in-the-loop planning and decision support. rakaposhi.eas.asu.edu/hilp-tutorial.
[34] Henry A. Kautz and James F. Allen. 1986. Generalized plan recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'86)*. 32–37.
[35] Emre Kıcıman and Matthew Richardson. 2015. 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*. ACM, 547–556.

[36] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444. DOI: https://doi.org/10.1038/nature14539

[37] Neal Lesh and Oren Etzioni. 1995. A sound and fast goal recognizer. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95)*. 1704–1710.

[38] Jonathan Lester, Tanzeem Choudhury, Nicky Kern, Gaetano Borriello, and Blake Hannaford. 2005. A hybrid discriminative/generative approach for modeling human activities. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'05)*. 766–772.

[39] Steven James Levine and Brian Charles Williams. 2018. Watching and acting together: Concurrent plan recognition and adaptation for human-robot teams. *J. Artif. Intell. Res.* 63 (2018), 281–359. DOI: https://doi.org/10.1613/jair.1.11243

[40] Lin Liao, Donald J. Patterson, Dieter Fox, and Henry A. Kautz. 2007. Learning and inferring transportation routines. *Artif. Intell.* 171, 5-6 (2007), 311–331.

[41] Lydia Manikonda, Tathagata Chakraborti, Sushovan De, Kartik Talamadupula, and Subbarao Kambhampati. 2014. AI-MIX: Using automated planning to steer human workers towards better crowdsourced plans. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*. 3004–3009.

[42] Jean Massardi, Mathieu Gravel, and Eric Beaudry. 2019. Error-tolerant anytime approach to plan recognition using a particle filter. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS'18)*. 284–291. Retrieved from https://aaai.org/ojs/index.php/ICAPS/article/view/3490.

[43] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS'13)*. 3111–3119.

[44] Reuth Mirsky and Ya'akov (Kobi) Gal. 2016. SLIM: Semi-lazy inference mechanism for plan recognition. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*. 394–400. Retrieved from http://www.ijcai.org/Abstract/16/063.

[45] Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *Proceedings of the International Workshop on Artificial Intelligence and Statistics*.

[46] Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. 2003. Autonomous helicopter flight via reinforcement learning. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS'03)*.

[47] Jeffrey Junfeng Pan, Qiang Yang, and Sinno Jialin Pan. 2007. Online co-localization in indoor wireless networks by dimension reduction. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'07)*. 1102–1107.

[48] Sinno Jialin Pan, James T. Kwok, Qiang Yang, and Jeffrey Junfeng Pan. 2007. Adaptive localization in a dynamic Wi-Fi environment through multi-view learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'07)*. 1108–1113.

[49] Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. 2014. How to construct deep recurrent neural networks. In *Proceedings of the 2nd International Conference on Learning Representations (ICLR'14)*. Retrieved from http://arxiv.org/abs/1312.6026.

[50] Ramon Fraga Pereira, Nir Oren, and Felipe Meneguzzi. 2017. Landmark-based heuristics for goal recognition. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*. 3622–3628. Retrieved from http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14666.

[51] Ramon Fraga Pereira, André Grahl Pereira, and Felipe Meneguzzi. 2019. Landmark-enhanced heuristics for goal recognition in incomplete domain models. In *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS'18)*. 329–337. Retrieved from https://aaai.org/ojs/index.php/ICAPS/article/view/3495.

[52] Charles J. Petrie. 1992. Constrained decision revision. In *Proceedings of the 10th National Conference on Artificial Intelligence*. 393–400.

[53] Martha E. Pollack, Laura E. Brown, Dirk Colbry, Colleen E. McCarthy, Cheryl Orosz, Bart Peintner, Sailesh Ramakrishnan, and Ioannis Tsamardinos. 2003. Autominder: An intelligent cognitive orthotic system for people with memory impairment. *Rob. Auton. Syst.* 44, 3-4 (2003), 273–282.

[54] Hangwei Qian, Sinno Jialin Pan, and Chunyan Miao. 2019. Distribution-based semi-supervised learning for activity recognition. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI'19), the 31st Innovative Applications of Artificial Intelligence Conference (IAAI'19), and the 9th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI'19)*. 7699–7706. Retrieved from https://aaai.org/ojs/index.php/AAAI/article/view/4765.

[55] Miquel Ramirez and Hector Geffner. 2009. Plan recognition as planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'09)*. 1778–1783.

[56] Miquel Ramirez and Hector Geffner. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'10)*. 1121–1126.

[57] Suchi Saria and Sridhar Mahadevan. 2004. Probabilistic plan recognition in multiagent systems. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'04)*.

[58] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. 2015. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Proceedings of the International Conference on Advances in Neural Information Processing Systems*. 802–810.

[59] Parag Singla and Raymond Mooney. 2011. Abductive Markov logic for plan recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'11)*. 1069–1075.

[60] Shirin Sohrabi, Michael Katz, Oktie Hassanzadeh, Octavian Udrea, and Mark D. Feblowitz. 2018. IBM scenario planning advisor: Plan recognition as AI planning in practice. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*. 5865–5867. DOI: https://doi.org/10.24963/ijcai.2018/864

[61] Shirin Sohrabi, Anton V. Riabov, and Octavian Udrea. 2016. Plan recognition as planning revisited. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'16)*. 3258–3264. Retrieved from http://www.ijcai.org/Abstract/16/461.

[62] Xin Tian, Hankz Hankui Zhuo, and Subbarao Kambhampati. 2016. Discovering underlying plans based on distributed representations of actions. In *Proceedings of the International Conference on Autonomous Agents & Multiagent Systems*. 1135–1143.

[63] Minh Do, Tuan Nguyen, Subbarao Kambhampati. 2011. Synthesizing robust plans under incomplete domain models. In *Proceedings of the AAAI Workshop on Generalized Planning*.

[64] Chunyu Xie, Ce Li, Baochang Zhang, Chen Chen, Jungong Han, and Jianzhuang Liu. 2018. Memory attention networks for skeleton-based action recognition. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*. 1639–1645. DOI: https://doi.org/10.24963/ijcai.2018/227

[65] Qiang Yang, Kangheng Wu, and Yunfei Jiang. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artif. Intell. J.* 171 (Feb. 2007), 107–143.

[66] Jie Yin, Dou Shen, Qiang Yang, and Ze-Nian Li. 2005. Activity recognition through goal-based segmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'05)*. 28–34.

[67] Jie Yin, Qiang Yang, and Lionel M. Ni. 2005. Adaptive temporal radio maps for indoor location estimation. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom'05)*. 85–94.

[68] Yu Zhang, Sarath Sreedharan, and Subbarao Kambhampati. 2015. Capability models and their applications in planning. In *Proceedings of the International Joint Conference on Autonomous Agents Multiagent Systems (AAMAS'15)*. 1151–1159.

[69] Yu Zhang, Sarath Sreedharan, Anagha Kulkarni, Tathagata Chakraborti, Hankz Hankui Zhuo, and Subbarao Kambhampati. 2017. Plan explicability and predictability for robot task planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'17)*. 1313–1320. DOI: https://doi.org/10.1109/ICRA.2017.7989155

[70] Vincent Wenchen Zheng and Qiang Yang. 2011. User-dependent aspect model for collaborative activity recognition. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'11)*. 2085–2090.

[71] Yonglei Zheng, Weng-Keen Wong, Xinze Guan, and Stewart Trost. 2013. Physical activity recognition from accelerometer data using a multi-scale ensemble method. In *Proceedings of the Conference on Innovative Applications of Artificial Intelligence (IAAI'13)*.

[72] Hankz Hankui Zhuo. 2017. Human-aware plan recognition. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence* 3686–3693. Retrieved from http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14936.

[73] Hankz Hankui Zhuo. 2019. Recognizing multi-agent plans when action models and team plans are both incomplete. *ACM Trans. Intell. Syst. Technol.* 10, 3 (2019), 30:1–30:24. DOI: https://doi.org/10.1145/3319403

[74] Hankz Hankui Zhuo and Subbarao Kambhampati. 2017. Model-lite planning: Case-based vs. model-based approaches. *Artif. Intell.* 246 (2017), 1–21.

[75] Hankz Hankui Zhuo and Lei Li. 2011. Multi-agent plan recognition with partial team traces and plan libraries. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'11)*. 484–489.

[76] Hankz Hankui Zhuo, Héctor Muñoz-Avila, and Qiang Yang. 2014. Learning hierarchical task network domains from partially observed plan traces. *Artif. Intell.* 212 (2014), 134–157. DOI: https://doi.org/10.1016/j.artint.2014.04.003

[77] Hankz Hankui Zhuo, Tuan Anh Nguyen, and Subbarao Kambhampati. 2013. Model-lite case-based planning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'13)*.

[78] Hankz Hankui Zhuo and Qiang Yang. 2014. Action-model acquisition for planning via transfer learning. *Artif. Intell.* 212 (2014), 80–103. DOI: https://doi.org/10.1016/j.artint.2014.03.004

[79] Hankz Hankui Zhuo, Qiang Yang, Derek Hao Hu, and Lei Li. 2010. Learning complex action models with quantifiers and implications. *Artif. Intell.* 174, 18 (2010), 1540–1569.

[80] Hankz Hankui Zhuo, Qiang Yang, and Subbarao Kambhampati. 2012. Action-model based multi-agent plan recognition. In *Proceedings of the International Conference on Neural Information Processing Systems (NIPS'12)*. 377–385.

Received December 2018; revised October 2019; accepted October 2019