CRADLE: An Online Plan Recognition Algorithm for Exploratory Domains

A

CRADLE: An Online Plan Recognition Algorithm for Exploratory Domains

REUTH MIRSKY, Dept. of Information Systems Engineering, Ben-Gurion University
YA'AKOV (KOBI) GAL, Dept. of Information Systems Engineering, Ben-Gurion University
STUART M. SHIEBER, Paulson School of Engineering and Applied Sciences, Harvard University

In exploratory domains, agents' behaviors include switching between activities, extraneous actions, and mistakes. Such settings are prevalent in real world applications such as interaction with open-ended software, collaborative office assistants, and integrated development environments. Despite the prevalence of such settings in the real world, there is scarce work in formalizing the connection between high-level goals and low-level behavior and inferring the former from the latter in these settings. We present a formal grammar for describing users' activities in such domains. We describe a new top-down plan recognition algorithm called CRADLE that uses this grammar to recognize agents' interactions in exploratory domains. We compare the performance of CRADLE with state-of-the-art plan recognition algorithms in several experimental settings consisting of real and simulated data. Our results show that CRADLE was able to output plans exponentially more quickly than the state-of-the-art without compromising its correctness, as determined by domain experts. Our approach can form the basis of future systems that use plan recognition to provide real-time support to users in a growing class of interesting and challenging domains.

CCS Concepts: •**Computing methodologies** → **Planning and scheduling;** *Distributed artificial intelligence; Modeling methodologies;*

1. INTRODUCTION

Exploratory domains are environments in which agents engage in behavior that includes switching between activities, extraneous actions, and mistakes [Amir and Gal 2013]. Open-ended educational software (such as one of the domains presented in this paper [Yaron et al. 2010]) is a paradigmatic example of such a setting, but other real-world environments, such as interactive drawing tools and integrated development environments (IDEs) exhibit similar characteristics.

Past work on plan recognition in exploratory domains operated off-line, assuming that all of the agents' actions are known at the time of recognition [Amir and Gal 2013; Uzan et al. 2015]. The focus of this paper is *on-line* plan recognition in exploratory domains, in which the recognition is being performed during the agents' interactions. The on-line variant of the recognition problem is more challenging because there is uncertainty over agents' future activities, so that explanations of possible future completions of agents' yet unseen activities given the observed actions must be maintained.

In this paper, we propose a novel approach for inferring users' activities in exploratory domains, which leads to significant improvement over the state of the art when evaluated on real data. Specifically, the contributions of this paper are as follows:

(1) A formal model for describing agents' activities in exploratory domains. The model explicitly captures partial plans and exogenous actions.
(2) An on-line plan recognition algorithm called CRADLE (Cumulative Recognition of Activities and Decreasing Load of Explanations), which extends the PHATT algorithm (Probabilistic Hostile Agent Task Tracker) [Geib and Goldman 2009] in the following way: It reduces the set of possible explanations on the fly using a set of domain-independent criteria, and updates arguments in the plan so that explanations remain consistent with new observations.[1]

[1]The term CRADLE is also evocative of the mechanical contrivance of the same name used in placer mining for washing out the gold-bearing soil, leaving only nuggets of gold.

(3) An empirical study showing CRADLE outperforms two state-of-the-art on-line plan recognition algorithms on a variety of data sets.

To demonstrate relative performance of the algorithms, we empirically evaluate them on three datasets from the literature containing logs of observable actions that exhibit aspects of exploratory behavior. We compare the performance of CRADLE to two state-of-the-art algorithms, PHATT [Geib and Goldman 2009] (which we extend to handle the types of plans in our datasets) and DOPLAR(Decision Oriented PLAn-Recognizer) [Kabanza et al. 2013] in terms of both correctness and speed. The results show that despite its inherent incompleteness, CRADLE performs well in practice, outperforming the state-of-the-art in accuracy and in solution time, in some cases achieving results exponentially faster.

The ability of CRADLE to deliver solutions in practical time means that the heuristic algorithm can provide correct solutions in more cases than algorithms that are complete in theory but prohibitively slow in practice. Our approach can form the basis of future systems for providing real-time support to users in a growing class of interesting and challenging domains.

2. RELATED WORK

We first describe works that use domain representations that are strictly less expressive than exploratory grammars. Bui [2003] used particle filtering to provide approximate solutions to on-line plan recognition problems. Avrahami-Zilberbrand and Kaminka [2005] handled temporal and free order constraints among actions by using plan libraries and recognize plans by traversing the tree in a manner that is temporally consistent with the observations, and making minimal commitments about matching actions to the grammar. This work was subsequently extended to provide an anytime expectation of time needed to recognize the plan [Fagundes et al. 2014] and to rank hypotheses based on the expected utility to the observer agent and probabilistic information in the grammar [Avrahami-Zilberbrand and Kaminka 2007]. Pynadath and Wellman [2000] developed a probabilistic grammar for modeling agents' plans that also included their beliefs about the environment but do not represent action parameters or exogenous actions. Blaylock and Allen [2004, 2006] developed an algorithm to infer the goal of users from their actions in a Linux shell environment. Their approach is probabilistic, and uses a chain of HMMs (Hidden Markov Models) to represent the possible paths an agent can take to perform a task. Actions in their domain representation include arguments, one of the characteristics of exploratory grammars. Additional approaches use probabilistic constraints over the plan duration [Duong et al. 2005] as well as resource dependencies [Sukthankar and Sycara 2008].

Geib and Goldman [2009] use and-or trees to represent domains and introduced a top-down plan recognition algorithm called PHATT which maintains a probability distribution over the hypothesis space. Exploratory grammars extend their representation to handling arguments and to be able to capture exogenous actions. A derivative of the PHATT algorithm called YAPPR (Yet Another Probabilistic Plan Recognizer) [Geib et al. 2008] is based on string writing and provides a more compact representation of the hypothesis space. This approach facilitates inference at the cost of representing only partial information about the agent's activities.

Some prior work in plan recognition focused on pruning the hypothesis space during run-time. The DOPLAR algorithm [Kabanza et al. 2013] refrains from generating plans that are not predicted to make significant contributions to the resulting hypothesis. This paper shows CRADLE was able to outperform the DOPLAR approach in the same empirical setting. Wiseman and Shieber [2014] propose an abduction technique that discriminatingly scores hypotheses based on features of the plan trees. These

works can predict the agent's goals and future actions but do not output a complete hierarchy of activities.

Lastly, we mention several works for recognizing students' interactions with e-learning software, which corresponds to two of the empirical settings used in the paper. Katz et al. [2007] used plan recognition algorithms to infer students' plans to solve problems in a simulated physics environment by comparing their actions to a set of predefined possible plans. Gal et al. [2012] proposed two algorithms for inferring students' plans in an exploratory setting consisting of a virtual laboratory for statistics education. This approach was extended to work with recursive grammars [Uzan et al. 2015]. These algorithms worked off-line and assumed that agents' complete interaction sequence is known at the time of recognition.

3. EXPLORATORY GRAMMARS

In this section we formalize a parameterized grammar and corresponding notions of (full and partial) plan trees, which are at the foundation of the CRADLE algorithm. This formalization captures the type of behaviors found in exploratory domains, namely free ordering and exogenous actions. Exploratory grammars can be seen as an extension of ID/LP grammars [Gazdar and Pullum 1981], which allows interleaving and an action to have arguments.

DEFINITION 1 (EXPLORATORY GRAMMARS). *An* exploratory grammar *is a tuple* $\langle \Sigma, N, G, U, V, P \rangle$ *where:*

— Σ *is a list of* primitive actions*,*
— N *is a list of* complex actions*,*
— *(*$A = \Sigma \cup N$ *is the set of* actions*.)*
— $G \subseteq N$ *is a set of* goal actions*,*
— U *is a set of* argument names*,*
— V *is a set of* argument values *that the named arguments can take on,*
— P *is a set of* production rules*, each of the form* $\langle \langle \alpha_0, \ldots, \alpha_n \rangle, LP, EC \rangle$*, where the following hold:*
  — $\alpha_0 \in N$ *and* $\langle \alpha_1, \ldots, \alpha_n \rangle \in A^*$*. This portion of the rule is conventionally notated* $\alpha_0 \rightarrow \alpha_1, \ldots, \alpha_n$*.*
  — LP *is a partial order capturing the* linear precedence *over the actions* $\alpha_1, \ldots \alpha_n$*, given as a set of pairwise inequations* $i \prec j$ *where* $1 \leq i, j \leq n$*. Intuitively,* $i \prec j$ *indicates that the action* α_i *should precede the action* α_j*.*
  — EC *is a set of* equational constraints *over argument values associated with the actions in the rule, notated as* $i.u = j.u'$ *or* $i.u = v$ *where* $u, u' \in U$ *and* $v \in V$ *and* $0 \leq i, j \leq n$*. A* 0 *index corresponds to the parent action, a non-zero index to the corresponding numbered child.*

(We will typically use α and its subscripted variants as metavariables over primitive or complex actions, and reserve β and its subscripted variants to range over primitive actions only.)

To illustrate these concepts we will use an open-ended educational software package for chemistry called VirtualLabs, which also comprises part of our empirical analysis. VirtualLabs allows students to design and carry out their own experiments for investigating chemical processes [Yaron et al. 2010] by simulating the conditions and effects that characterize scientific inquiry in the physical laboratory. We use a problem called "Oracle" as a running example:

> Given four substances A, B, C, and D that react in a way that is unknown, design and perform virtual lab experiments to determine which of these substances react, including their stoichiometric coefficients.

$$
\begin{aligned}
&\textit{SAME} \rightarrow \textit{SAME}, \textit{SAME} \\
&\qquad LP = \{1 \prec 2\} \qquad (1)\\
&\qquad EC = \{0.d = 1.d, 0.d = 2.d\} \\
&\textit{INTER} \rightarrow \textit{SAME}, \textit{SAME} \\
&\qquad LP = \{1 \prec 2\} \qquad (2)\\
&\qquad EC = \{0.s = 1.s, 1.d = 2.s, 0.d = 2.d\} \\
&\textit{SAME} \rightarrow \textit{INTER} \\
&\qquad LP = \emptyset \qquad (3)\\
&\qquad EC = \{0.s = 1.s, 0.d = 1.d\} \\
&\textit{SAME} \rightarrow \textit{pour} \\
&\qquad LP = \emptyset \qquad (4)\\
&\qquad EC = \{0.s = 1.s, 0.d = 1.d\}
\end{aligned}
$$

Fig. 1. Grammar for VirtualLabs domain

(All questions used in the two test domains are provided in Appendix A.)

We define an exploratory grammar for the VirtualLabs software as follows, adapting the rules suggested initially by Amir and Gal [2013]. The set of primitive actions Σ represents rudimentary operations with VirtualLabs that cannot be decomposed. These include a single action *pour* that represents pouring a chemical substance from a source flask to a destination flask. Since the source and destination represent arguments of the action, we encode them with two arguments in U, s and d respectively. The set of complex actions N describe composite activities such as mixing two compounds together. The set N in the grammar includes the following two types of activities: The first activity, the complex action *SAME*, denotes a pour from two source flasks into a single destination flask. We call this a "same destination" activity. The second activity, denoted by the complex action *INTER*, represents a pour from a source flask to a destination flask via an intermediate flask. We call this an "intermediate flask" activity. The set of argument names U and values V in the grammar identify flasks and their contents. The set of goals G in the grammar is comprised of both complex actions *SAME* and *INTER*. The rules in the grammar are shown in Figure 1.

Rule (1) describes the recipe for achieving the complex action *SAME* by combining two *SAME* actions. The EC constraint of this rule requires that both pours end up in the same destination flask identified as $0.d$ and the LP constraint requires that the pours occur in sequence ($1 \prec 2$). The rule in Equation (2) describes the recipe for achieving the complex action *INTER* by combining two *SAME* actions. The EC constraint of this rule requires that the destination flask of the first pour is the source flask of the second pour ($1.d = 2.s$). The rule in Equation (3) allows converting *INTER* actions to *SAME* actions (to reduce the number of rules required). The rule in Equation (4) grounds the complex action *SAME* in the primitive pour action *pour*.

A key construct in the paper are *partial plan trees*, which are used to describe agents' possibly incomplete and interleaving activities. Before formalizing this notion we need to make the following definitions. A *binding* is a partial function from argument names U to values V. For example, we will notate a binding D that maps the argument s to value 1 and d to 2 as $\{s = 1, d = 2\}$ and use the notation $D[s]$ for the value of the binding D at argument s (that is, the value 1), implicitly requiring that the binding be defined at that argument. A *grounded action* is a pair $\langle \alpha, D \rangle$ of an action $\alpha \in A$ and a

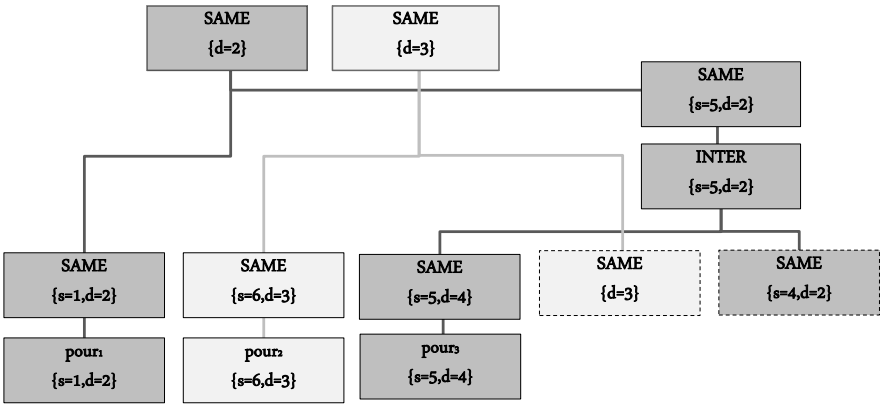


Fig. 2. Two plan trees comprising the explanation for observation sequence $\langle \textit{pour}_1, \{s = 1, d = 2\}\rangle, \langle \textit{pour}_2, \{s = 6, d = 3\}\rangle, \langle \textit{pour}_3, \{s = 5, d = 4\}\rangle$.

binding D. An *observation sequence* is a sequence $\langle\langle \beta_1, D_1\rangle, \ldots, \langle \beta_n, D_n\rangle\rangle$ of grounded primitive actions (that is, $\beta_i \in \Sigma$ for all i).

An example of an observation sequence in VirtualLabs is presented below. It includes a sequence of basic *pour* actions (including values for source and destination arguments in the corresponding bindings), which are indexed for convenience according to their temporal order of execution:

$$\langle\langle \textit{pour}_1, \{s = 1, d = 2\}\rangle, \langle \textit{pour}_2, \{s = 6, d = 3\}\rangle, \langle \textit{pour}_3, \{s = 5, d = 4\}\rangle\rangle$$

A *partial plan tree* is an ordered tree where each node is labeled with a grounded action. We refer to the set of all leaves of a tree as the *frontier* of the tree. Notice that using this definition, a plan tree's frontier can contain both basic and complex actions. A partial plan tree whose frontier is composed solely of primitive actions is a *full plan tree*. Henceforth, we use the term *plan tree* for partial plan trees. Plan trees are well formed if they obey all the kinds of constraints in the grammar: The immediate dominance (ID) relationships between each parent and its children are consistent with a production rule; the ordering of children is consistent with the linear precedence (LP) rules; and the argument values are consistent with the equational constraints (EC). We define appropriate checks for ID and EC acceptability next, postponing LP acceptability until presenting the context of observation sequences that ground the orderings.

DEFINITION 2 (ID-ACCEPTABILITY). *A parent node $\langle \alpha_0, D_0\rangle$ in a plan tree with children nodes $\langle \alpha_1, D_1\rangle, \ldots, \langle \alpha_n, D_n\rangle$ is* ID-acceptable *if there is a rule $p = \langle \alpha_0 \rightarrow \alpha_1, \ldots, \alpha_n, LP, EC\rangle \in P$. In such a case, we say that the rule p* sanctions *the node.*

DEFINITION 3 (EC-ACCEPTABILITY). *An ID-acceptable node is called* EC-acceptable *if:*

(*1*) *For every $i.u = j.v$ in EC of the rule that sanctions the node, we have that if either $D_i[u]$ or $D_j[v]$ is defined then $D_i[u] = D_j[v]$.*
(*2*) *For every $u.i = v$ in EC that sanctions the node, we have that $D_i[u]$ is defined and $D_i[u] = v$.*

We extend these definitions to say a plan tree is ID-acceptable if each node in the plan tree is ID-acceptable, and similarly for EC-acceptability.

To illustrate, consider the plan trees in Figure 2. The plan tree with the root labeled $\langle \textit{SAME}, \{d = 2\}\rangle$ is ID-acceptable because the rule $\textit{SAME} \rightarrow \textit{SAME}, \textit{SAME}$ in Equation (1) sanctions the root node; the rule $\textit{SAME} \rightarrow \textit{INTER}$ in Equation (3) sanctions the node $\langle \textit{SAME}, \{s = 5, d = 2\}\rangle$; the rule $\textit{SAME} \rightarrow \textit{pour}$ in Equation (4) sanctions the node $\langle \textit{SAME}, \{s = 1, d = 2\}\rangle$, and so forth. This plan tree is EC-acceptable, because the EC constraints of the rule that sanctions each node are satisfied by its children nodes. For example, we have that the value $d = 2$ holds for the root and all its children nodes. Similarly, it can be shown that the plan tree with root labeled $\langle \textit{SAME}, \{d = 3\}\rangle$ is ID- and EC-acceptable. We will make crucial use of the fact that partial plan trees do not have to extend down to primitive actions, and in fact, there may be no way to extend any given partial plan tree to a full plan tree.

Exploratory grammars allow actions in a plan tree to be observed in free order in an observation sequence matching the plan tree, provided that the nodes (and their descendants, down to the observations) meet the declared LP constraints. We define this condition formally:

DEFINITION 4 (LP-ACCEPTABILITY). *An observation sequence* $\langle\langle \beta_1, D_1\rangle, \ldots, \langle \beta_n, D_n\rangle\rangle$ *is* LP-acceptable *with respect to an ID-acceptable plan tree with frontier* $\langle \gamma_1, E_1\rangle, \ldots, \langle \gamma_k, E_k\rangle, (n \leq k)$ *if the following holds:*

(*1*) *There exists a one-to-one (but perhaps not onto) function* π *from* $\{1, \ldots, n\}$ *to* $\{1, \ldots, k\}$ *such that* $\beta_i = \gamma_{\pi_i}$ *and* $D_i = E_{\pi_i}$.
(*2*) *For each node* $\langle \alpha_0, D\rangle$ *in the plan tree with children* $\langle\langle \alpha_1, D_1\rangle, \ldots, \langle \alpha_n, D_n\rangle\rangle$*, sanctioned by the rule* $p = \langle \alpha_0, \rightarrow \alpha_1, \ldots, \alpha_n, LP, EC\rangle$*, and for each inequation* $i \prec j \in LP$*, let* Y_i *and* Y_j *be the set of leaf nodes in the frontier of the tree that are descendants of the node* α_i *and* α_j*, respectively. For all* $\gamma_{\pi_l} \in Y_i$ *and* $\gamma_{\pi_m} \in Y_j$*, it must hold that* $l < m$ *for* β_l *and* β_m.

Intuitively, the first condition provides a mapping from the observations to the plan frontier. The last condition states that the ordering over the leaves in the plan tree must not conflict with the ordering constraints defined in the rules that make up the plan tree.

To illustrate, in Figure 2 the observation sequence $\langle \textit{pour}_1, \{s = 1, d = 2\}\rangle$ and $\langle \textit{pour}_3, \{s = 5, d = 4\}\rangle$ is LP-acceptable with respect to the plan tree with the root labeled $\langle \textit{SAME}, \{d = 2\}\rangle$. To see this consider the permutation over observations that match $\langle \textit{pour}_1, \{s = 1, d = 2\}\rangle$ and $\langle \textit{pour}_3, \{s = 5, d = 4\}\rangle$ with their corresponding leaf nodes in the plan tree. The LP constraint $1 \prec 2$ of the rule that sanctions the node is satisfied in that $\textit{pour}_1$ precedes $\textit{pour}_3$ in the tree. The frontier of this tree includes a node $\langle \textit{SAME}, \{s = 4, d = 2\}\rangle$ in dashed outline which is not labeled with a terminal action (and which does not conflict with the LP-acceptability of the observation sequence.) This node is called an "open frontier" and is a place-holder for a yet unseen activity that is expected to occur in the future. It will play a central role in the recognition algorithm that is described in the next section.

We are now ready to state formally how observation sequences are accepted by the grammar.

DEFINITION 5 (ACCEPTING OBSERVATIONS). *An observation sequence* $\langle\langle \beta_1, D_1\rangle, \ldots, \langle \beta_n, D_n\rangle\rangle$ *is* accepted *by the grammar if there exists an ID- and EC-acceptable plan tree such that*

(*1*) *The observation sequence is LP-acceptable with respect to the plan tree; and*
(*2*) *The root of the tree is of the form* $\langle \alpha_0, D_0\rangle$ *and* α_0 *is a goal in the grammar, that is,* $\alpha_0 \in G$.

For example, in Figure 2, the observation sequence $\{\textit{pour}_1, \textit{pour}_3\}$ is accepted by the grammar using the plan tree that is rooted in $\langle \textit{pour}, \{d = 2\}\rangle$ and the observation sequence $\{\textit{pour}_2\}$ is accepted by the grammar using the the plan tree with the root labeled $\langle \textit{SAME}, \{d = 3\}\rangle$. Note that both of these plan trees are incomplete, in that they have actions that are not grounded.

Lastly, we define the notion of an explanation, which is used to describe activities in the grammar as a union of plan trees. Allowing explanations to include several plan trees captures trial-and-error and interleaving plans, both of which are endemic to exploratory domains.

DEFINITION 6 (EXPLANATION). *An* explanation *of an observation sequence is a partition of the sequence into subsets of observation sequences that are accepted by the grammar. We then say the observation sequence is "explained" by the grammar.*

Figure 2 shows an observation sequence $\{\langle \textit{pour}_1, \{s = 1, d = 2\}\rangle, \langle \textit{pour}_2, \{s = 6, d = 2\}\rangle, \langle \textit{pour}_3, \{s = 5, d = 4\}\rangle\}$ that is explained by the grammar. The observations are presented in order of execution, from left to right. The explanation consists of two interleaving plan-trees.

4. THE CRADLE ALGORITHM

CRADLE[2] is a top-down probabilistic plan recognition algorithm that extends the PHATT algorithm of Geib and Goldman [2009] to exploratory domains. It receives as input an exploratory grammar and an observation sequence. It outputs a set of explanations for the observation sequence according to the grammar.

We begin by adapting some necessary definitions from PHATT to our setting. We first define the notion of a *leftmost* child in a plan tree as one that is allowed to be first among its siblings given the LP constraints of the rule that sanctions its parent.

DEFINITION 7 (LEFT-MOST CHILD). *Let T be a plan tree and let $\langle \alpha_0, D_0\rangle$ be a parent node in T with children $\langle\langle \alpha_1, D_1\rangle, \ldots, \langle \alpha_i, D_i\rangle, \ldots, \langle \alpha_n, D_n\rangle\rangle$. We say $\langle \alpha_i, D_i\rangle$ is a* left-most child *of node $\langle \alpha_0, D_0\rangle$ if*

(*1*) *$\langle \alpha_0, D_0\rangle$ is ID-acceptable and sanctioned by rule p with linear precedence constraints LP; and*
(*2*) *For each child node labeled $\langle \alpha_j, D_j\rangle$ such that $j \neq i$, it holds that $j \prec i \notin LP$.*

We lift the notion of leftmost child to a notion of *left-most tree*, which incorporates a single primitive leftmost child, and which we will use to incorporate new observations into existing plan trees.

DEFINITION 8 (LEFT-MOST TREE). *A* left-most tree *deriving an observation $\sigma = \langle \beta_i, D_i\rangle$ is an ID- and EC-acceptable plan tree with frontier that includes $\langle \beta_i, D_i\rangle$ such that*

(*1*) *$\beta_i \in \Sigma$ for all $j \neq i$; and*
(*2*) *For any child node labeled $\langle \alpha_j, D_j\rangle$ and its parent $\langle \alpha_k, D_k\rangle$ in the path from $\langle \beta_i, D_i\rangle$ to the root, the node $\langle \alpha_j, D_j\rangle$ is a left-most child of $\langle \alpha_k, D_k\rangle$.*

The node labeled with $\langle \beta_i, D_i\rangle$ is called the *left corner* of the tree.[3] Any leaf in a left-most tree that is not the left corner is an *open-frontier* node. Such nodes represent place holders for possible future activities that depend on observations that have yet to be seen.

[2]The code for CRADLE is freely available at https://github.com/ReuthMirsky/CRADLE
[3]In the PHATT approach, this node is called the "foot" of the tree.

The *generating set* of a primitive action σ is the set of all left-most trees in the grammar that derive σ. Suppose we extend the observation sequence $\langle\{\textit{pour}_1, \textit{pour}_2, \textit{pour}_3\}\rangle$ with a new observation $\langle \textit{pour}_4, \{s = 7, d = 3\}\rangle$. Figure 3 shows the generating set of the observation $\textit{pour}_4$, which consists of three trees: Figure 3(a) shows a tree that derives $\textit{pour}_4$ using the base-case rule $\textit{SAME} \rightarrow \textit{pour}$ in the grammar; Figure 3(b) shows a left-most tree that derives $\textit{pour}_4$ by composing the same-destination rule $\textit{SAME} \rightarrow \textit{SAME}, \textit{SAME}$ with the base-case rule; Figure 3(c) shows a left-most tree that derives $\textit{pour}_4$ by composing the intermediate-flask rule in $\textit{INTER} \rightarrow \textit{SAME}, \textit{SAME}$ with the base-case rule. In all trees, leaves in dashed outlines are open-frontier nodes. Importantly, for recursive grammars, such as the one of Figure 1, the number of trees in the generating set is potentially unbounded. In practice, we limit the depth of recursion in trees in the generating set to one. This is the maximum number of times that each rule is allowed to be used recursively for deriving an observation.

Any of these trees from the generating set could be "spliced in" to the partial plan trees shown in Figure 2 to extend the coverage of those trees to explain the additional $\textit{pour}_4$ observation. Alternatively, any of these trees might stand alone as a partial plan tree to augment the current explanation of the observation sequence with a new separate subsequence.

The CRADLE algorithm, whose main methods are shown in Figure 4, uses just such kinds of operations – maintaining a set of heuristically pruned nondeterministic possibilities – to construct a set of explanations E for an observation sequence $\sigma_1, \ldots, \sigma_n$ incrementally from left to right. At any given point in the algorithm, a set E_{t-1} holds the set of explanations for the subsequence of observations $\sigma_1, \ldots, \sigma_{t-1}$, which is to be augmented with the next observation σ_t.

The function ADDOBS is responsible for adding a new observation $\sigma_t = \langle \beta_t, D_t \rangle$ into the existing set of explanations E_{t-1} given a set of filtering rules F. (The role of filtering rules is explained in Section 4.1.) The function proceeds in two steps. First, it attempts to incorporate the observation into an existing plan tree in an explanation $e \in E_{t-1}$ (line 4 in Figure 4) by replacing an open-frontier node in one of the plan trees in e with a left-most tree from the generating set that derives σ_t, an operation performed by the function COMBINEINEXPLANATION. Each time an explanation is modified it is added to the set of possible explanations (line 5 in Figure 4).

Second, it adds a new plan tree to an existing explanation, an operation performed by ADDNEWTREE, which attempts to add each plan-tree T_σ in the generating set of σ_t as a new plan-tree in e if the root of T_σ is in the goal set G of the grammar.

CRADLE's probability model is similar to the one used by PHATT, computing the probability of each explanation as the conditional probability that this explanation is what the agent is really doing, given the sequence of observations. More formally, for each explanation exp for the observation sequence obs, we need to compute $P(exp \wedge obs) = P(exp) \cdot P(obs \mid exp)$. Geib and Goldman [2009] show that this computation can be decomposed to $P(exp \wedge obs) = P(goals) \cdot P(plans \mid goals) \cdot P(obs \mid exp)$, where $P(goals)$ is the probability that the agent's goals are exactly the root nodes of the plan trees in exp, $P(plans \mid goals)$ is the probability that the specific plan trees of the explanations are the way to achieve the root goals and $P(obs \mid exp)$ is the probability that the observed actions are chosen from all possible actions that could have been taken to extend exp after every step.

4.1. Extensions over PHATT algorithm

The CRADLE algorithm extends the PHATT approach [Geib and Goldman 2009] in three ways: First, it supports exploratory grammars (Definition 1), meaning that CRADLE receives as input an exploratory grammar and an observation sequence, and outputs a set of hypotheses, each of which is an explanation of the observation sequence in

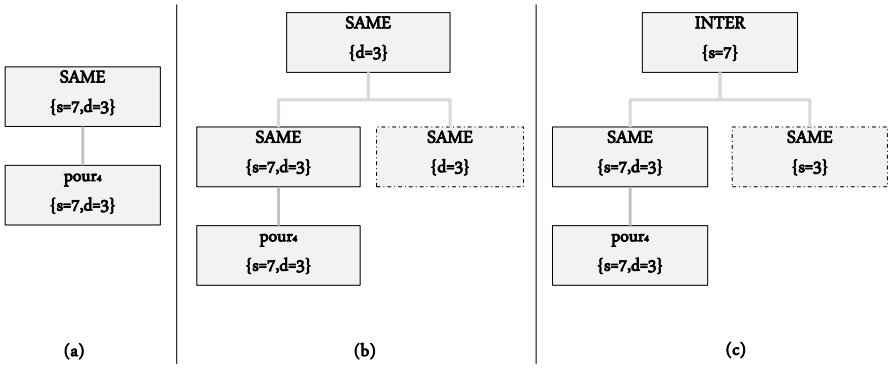


Fig. 3. The generating set of the observation $pour_4$

the sense of Definition 6. Second, it filters redundant explanations according to a set of domain-independent conditions. A filter is a function taking a candidate explanation e, and returning *true* or *false* depending on whether the candidate explanation does or does not pass a certain condition. Third, CRADLE can handle exogenous actions and mistakes and can omit them from the set of explanations as needed. We expand on each of these extensions in turn.

4.1.1. Incorporating new observations. Integrating new observations requires careful bookkeeping to maintain the consistency of explanations in CRADLE. First, the argument values for uninstantiated actions in the plan tree need to be updated to reflect the new information that is embedded in the plan. Second, a plan deriving the new observation can replace a node in an existing explanation only if the new plan tree is ID-acceptable and satisfies all of the LP and EC constraints that are embedded in the rules used to construct the plan. For example, consider the plan tree in Figure 5. Adding a new observation $\langle pour_3, \{s = 2, d = 3\}\rangle$ to the plan tree requires propagating the value $d = 3$ to the open-frontier nodes in the plan tree in which the value of d is not instantiated, as is shown in the figure.

In order to maintain this consistency, the SUBSTITUTENODE(T, T_σ, o) function in Figure 4 clones the original plan tree T and substitutes the node representing the open frontier action o with the plan tree T_σ that explains the new observation σ. The UPDATE function recursively propagates the argument values from the root of a subtree in the plan to the rest of the plan tree. Specifically, if the arguments associated with a node n and its parent violate a constraint in the rule that derives n and its siblings, then UPDATE returns false, and SUBSTITUTENODE does not incorporate σ into the plan tree T. The result of a successful substitution is a modified plan-tree T that is ID- and EC-acceptable and a new explanation for the observation sequence up to time t.

To illustrate, Figure 6 shows two ways of incorporating observation $pour_4$ into the explanations originally shown in Figure 2. In Figure 6(a), the plan-tree in the generating set of Figure 3(a) is substituted for the open frontier node $\langle \textit{SAME}, \{d = 3\}\rangle$. In Figure 6(b), the plan-tree in the generating set of Figure 3(b) is substituted for the same open frontier node. In total, incorporating the observation $pour_4$ into the existing explanation set results in five possible explanations for the observation sequence $pour_1, \ldots, pour_4$.

```
1: function AddObs(E_{t-1}, σ_t, F, R)
2:     for all explanation e ∈ E_{t-1} do
3:         E_t ← ∅
4:         E_t ← E_t ∪ CombineInExplanation(e, σ_t)
5:         E_t ← E_t ∪ AddNewTree(e, σ_t)
6:     E_t ← Filter(E_t, F)
7:     if | E_t | < R then
8:         E_t ← E_t ∪ E_{t-1}
9:     return E_t

10: function CombineInExplanation(e, σ)
11:     E ← ∅
12:     for all plan T ∈ e do
13:         for all open frontier item o ∈ T do
14:             for all plan tree T_σ in generating set of σ do
15:                 T' ← SubstituteNode(T, T_σ, o)
16:                 if T' is a valid tree then
17:                     e' ← e.replaceTree(T, T')
18:                     E ← E ∪ e'
19:     Return E
20:
21: function AddNewTree(e, σ)
22:     E ← ∅
23:     for all T_σ a leftmost tree deriving σ do
24:         if root(T_σ) ∈ G then
25:             e ← e ∪ T_σ
26:             E ← E ∪ e
27:     return E

28: function Filter(E, F)
29:     E' ← ∅
30:     for all explanation e ∈ E do
31:         for all filter f ∈ F do
32:             if f(e, E) then
33:                 E' ← E' ∪ e
34:     return E'

35: function SubstituteNode(T, T_σ, o)
36:     T' ← Clone(T)
37:     replace node o in T' with T_σ.
38:     for all node n in path from o to root(T') do
39:         if Update(parent(n), n)==false then
40:             return T
41:     return T'
```

Fig. 4. The CRADLE algorithm

4.1.2. Filtering the set of explanations. The following filters used by CRADLE heuristically prune the space of possible explanations based on certain thresholds:

— The *aging* filter prefers explanations in which successive observations extend existing sub-plans in the explanation rather than generate new plans. It discards explanations in which observations have not extended an existing plan for a given number of iterations.

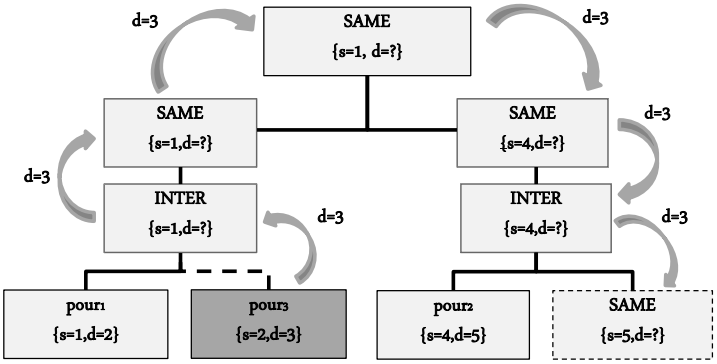


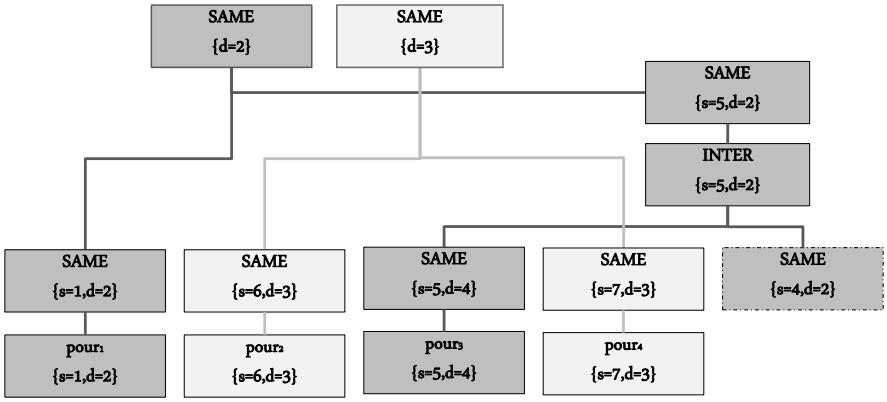
Fig. 5. Maintaining consistency of explanations

— The *frontier size* filter prefers explanations that make fewer commitments about future observations. It measures the number of open items in the frontier of each explanation, discarding explanations where this count is more than τ_f standard deviations above the mean number of open-frontier nodes per explanation in the current set of explanations.
— The *explanation size* filter prefers explanations with a smaller number of plan trees. It discards explanations in which the number of plans is more than τ_e standard deviations above the mean number of plans in the current set of explanations.
— The *probability* filter prefers explanations with a higher likelihood. As in PHATT, we compute the probability of each explanation as the product of the probabilities assigned to each rule that was used to make up the explanations given the observation sequence. The filter discards explanations whose probability of generating the observation sequence is within τ_p standard deviations below the mean probability.

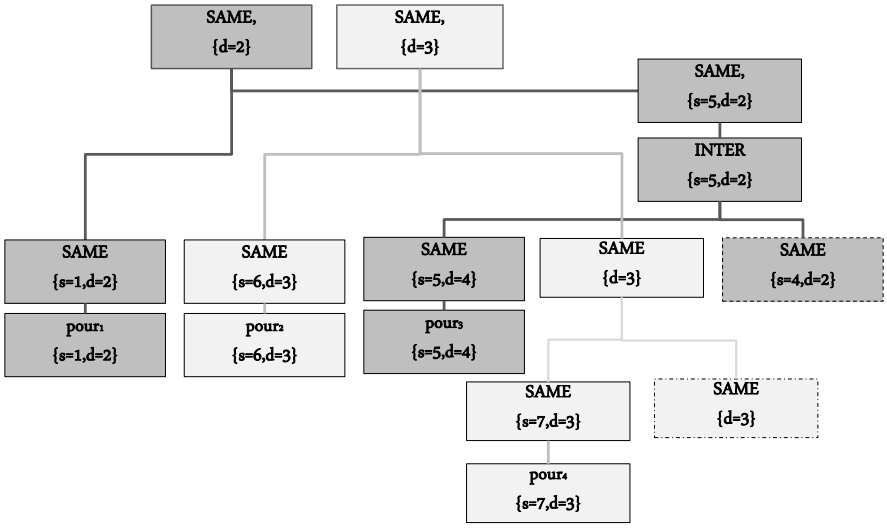
For the frontier size, explanation size, and probability filters we used a relative threshold value (the distance from the mean). We hypothesized that by using a relative threshold value we would be able to adapt the filter behavior to specific instances. For example, when the mean explanation size is high, the threshold value for the explanation size filter would increase, and conversely when the mean explanation size is low. We experimented with several possible values for the thresholds τ_f, τ_e, τ_p on a set of held out instances, eventually choosing a value of zero for all these measures. Thus, the threshold value for the explanation size filter was set to the mean number of explanations (and similarly for the frontier size and probability filters).

Line 6 in the ADDOBS function in Figure 4 applies the different filter conditions. The filter functions also have access to E_t so that they may use statistics over E_t as thresholds. To illustrate, after adding the observation $pour_4$ to the existing set of explanations, three of the five resulting explanations will be filtered by the explanation size filter. Two possible explanations remain, shown in Figures 6(a) and 6(b).

4.1.3. Exogenous actions. Lastly, we explain how CRADLE deals with exogenous actions, actions that do not form a necessary part of a plan and represent mistakes or explorations. Such actions may be impossible to incorporate into any existing explanation without compromising at least a single constraint, so they need special handling. Exogenous actions are identified when both functions COMBINEINEXPLANATION and ADDNEWTREE in the CRADLE algorithm return the empty set, because they cannot combine an observation to the existing explanation set. A threshold parameter R



(a)



(b)

Fig. 6. Updated explanation set after observing action $\langle pour_4, \{s = 7, d = 3\}\rangle$

determines the minimal number of explanations required to be able to combine the observation so that it is not considered exogenous. CRADLE explicitly reasons about the possibility that an action is exogenous, and includes explanations in which this action is omitted. Specifically, the ADDOBS function keeps count of the number of times that action σ_t was successfully incorporated into an existing explanation. If this number falls below a certain threshold R then CRADLE also considers explanations E_{t-1} which do not include the action σ_t (lines 7–8).

The three main functionalities of CRADLE are modular, in that they can be turned "on" or "off" independently. In fact, when the filtering, argument values, and exogenous actions aspects are dropped, we have essentially the PHATT algorithm.

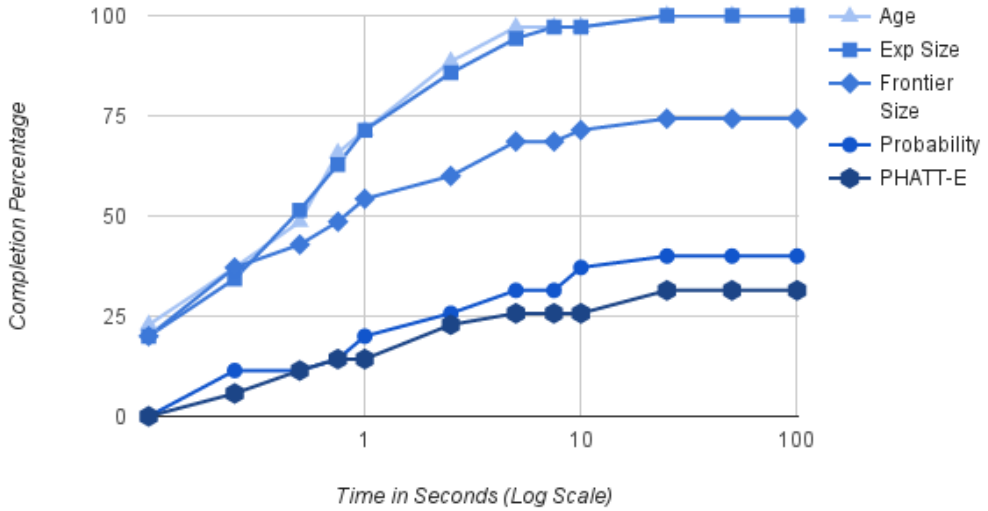


Fig. 7. Solution percentage as a function of run-time for the extended PHATT algorithm and several CRADLE variants

5. EMPIRICAL EVALUATION

Despite the inherent incompleteness from using the filtering methods to prune the space of explanations, the CRADLE algorithm performs well in practice, both in run-time and in accuracy, as we show empirically in experiments with three domains.

5.1. The VirtualLabs domain

The first domain involves students' interactions with the VirtualLabs system when solving two different types of problems: the Oracle problem described in Section 3, and a problem called "Unknown Acid" which required students to determine the concentration level of an unknown acid solution by performing a chemical titration process (see Apendix). We sampled 35 logs of students' interactions with VirtualLabs to solve the above problems (20 Oracle and 15 Unknown Acid instances). These two problems differ widely in the types of solution strategies they require from students, which is reflected in the length and the types of actions in the logs that we sampled. The logs consisted of between 4 and 211 actions. We ran different versions of the CRADLE algorithm, one for each filter variant (age, explanation size, frontier size and probability). We set the exogenous-action threshold R to 0, meaning that an action will be considered exogenous only if it cannot be included in any tree in any explanation.

All actions in the VirtualLabs domain include arguments and values, which are not supported by the original PHATT approach. Therefore, we used an augmented version of the PHATT algorithm with arguments and consistency checking that we will call PHATT-E. We ran both PHATT-E and the CRADLE variants on the VirtualLabs logs, assigning a uniform probability distribution over the rules in the grammar.

5.1.1. Speed and termination. Figure 7 shows the performance obtained using PHATT-E and CRADLE with the combined filter criteria, running both algorithms on a commod-

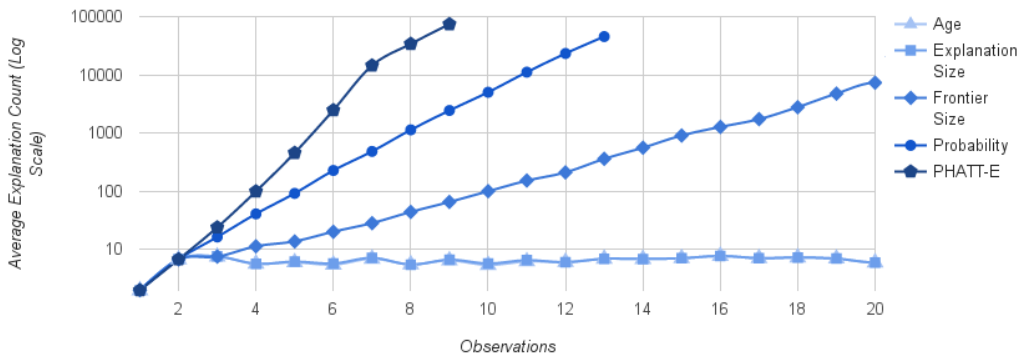


Fig. 8. Number of generated explanations

ity core-i7 computer. The log actions were fed to each algorithm step-by-step to simulate real-time conditions. The x-axis in the figure corresponds to time (in seconds). The y-axis measures the percentage of problems solved.

The figure shows that for each filter variant, the CRADLE algorithm was able to outperform the PHATT-E approach. The best performance for CRADLE was achieved using the AGE and Explanation Size filter which were able to complete almost all of the logs in less than 10 seconds of CPU time. In contrast, PHATT-E was not able to complete more than 30% of the logs in a designated time frame of 100 seconds of CPU time, which is a conservative upper bound for on-line recognition, in which feedback needs to be provided quickly to a user. There was no single filter method that outperformed all of the other methods for all logs in this domain. Also, combining the full set of filters did not improve the average performance of the methods over the best individual filter.

Figure 8 compares the average number of explanations generated by CRADLE and PHATT-E for different number of observations. For each number of observations, we only show cases in which there were at least 15 logs of at least the given length. In practice, this limited the number of observations shown in the graph to 20, as shown in the x-axis in the graph. As shown in the figure, the average number of explanations maintained by PHATT-E grows exponentially in the number of observations. By way of example, PHATT-E generated 142 different explanations for one of the logs with 4 observations, and more than 10,000 explanations for one of the logs with 8 observations. The PHATT-E algorithm was not able to terminate in the designated 100 seconds of CPU time on logs with more than 8 observations. The best performance was attributed to the CRADLE variants using the Explanation- and Age-Size filters, which were able to keep the size of the explanation set relatively constant. Interestingly, the probability filter was not able to terminate on logs over 13 observations. We attribute this to the fact that the variance over the probability of explanations was low, such that the probability filter was not able to prune a substantial number of explanations.

Next, we compare the run-time of several CRADLE variants with different explanation set size growth rates (frontier size and explanation size filters) and PHATT-E for the different logs containing 20 observations or less. We chose these two filters as they show different trade-offs between runtime and the number of generated explanations. Figure 9 shows the average run-time on different log sizes, measured in seconds, presented in a logarithmic scale. The average run-time of PHATT-E grows exponentially

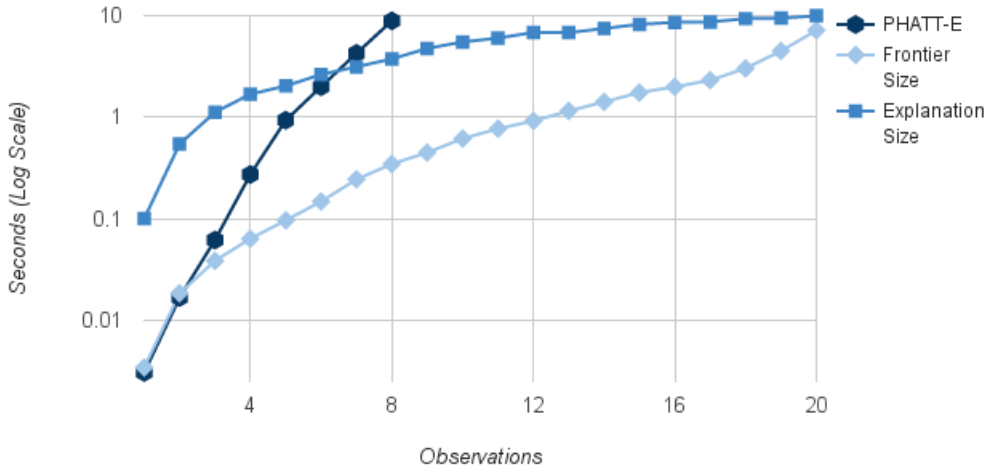
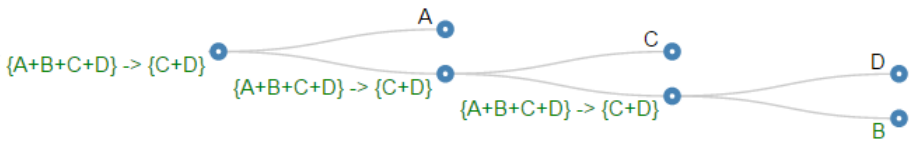


Fig. 9. Runtime of PHATT-E and CRADLE variants with frontier and explanation size filters

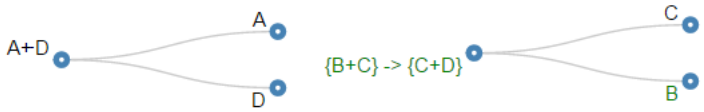
with the log size. Both CRADLE variants completed the runs in less than 10 seconds of CPU time for all of the logs, with the frontier size filter constantly outperforming the explanation size filter. For a small number of observations, the number of generated explanations is relatively low, and PHATT-E was able to terminate more quickly than the CRADLE variants. However, once the number of observations grows, the runtime of PHATT-E increases substantially, due to the overhead of maintaining the explanation set.

Lastly, we note that in the VirtualLabs domain there is no possibility for an action to be exogenous, since all actions can be the first action in a new tree using the AddNewTree function. Indeed, the VirtualLabs grammar shown in Figure 1 describes how students perform chemistry activities using the software (e.g., pouring a compound from flask A to B). It does not describe students' higher level solution strategies (such as comparing each pair of compounds in order to determine which are the reactants in the Oracle example of Section 3.) Gal et al. [2015] show how to infer such higher-level strategies by showing plans of students' activities as generated by a plan recognition algorithm to teachers.

5.1.2. Agreement with domain experts. Given that CRADLE is able to terminate in reasonable time on the instances we collected, the next question to ask is whether the solutions it outputs make sense in practice according to a domain expert. We used plans generated from partial logs when comparing with domain experts' opinions, effectively simulating an on line setting. To this end, we cut the 35 logs to the maximal size for which PHATT-E was able to terminate within the designated 100-second runtime. We compared the performances of PHATT-E and CRADLE with all filters combined. We selected the explanation to visualize with highest probability among all those that did not include any open frontier nodes (that is, they provided a full description of the log). If no explanation met this criterion, we chose the explanation with the highest probability. In practice, 23 out of the 35 logs had at least one explanation with no open frontier.



(a) CRADLE



(b) PHATT-E

Fig. 10. Plan Visualization

We engaged a set of domain experts to evaluate the quality of generated solutions for these logs. All of the experts were chemistry education researchers with prior experience working with VirtualLabs and similar software in classes. Each expert was shown a pair of visualizations for each log, one generated by each algorithm. They were asked to indicate which of the visualizations was correct, the one generated by CRADLE, by PHATT-E, by both, or by neither of the algorithms. (The identity of the algorithms was not disclosed.) Figure 10(a) and Figure 10(b) show the explanations that were generated by CRADLE and PHATT-E respectively on one of the ORACLE logs used in the study. An explanation was visualized to the experts using a dedicated GUI as a set of trees in which nodes representing complex actions were labeled with information about the chemical reactions that occurred in the activity represented by the complex action. Nodes representing basic actions were labeled with observations. For example, the root node labeled $A + B + C + D \longrightarrow C + D$ represents an activity of mixing four solutions together which resulted in a chemical reaction that consumed substances A and B and generated substance D. A data panel (not shown in the figure) shows the amounts of each chemical used and obtained in the student's interaction. All of the experts preferred the CRADLE explanation over the PHATT-E explanation in the log instance that relates to the example. In total, the domain experts evaluated 105 log instances from the two problems. The explanation generated by CRADLE was rated at least as highly as the one generated by PHATT-E in 100 instances (95%), and rated strictly better than the explanations outputted by PHATT-E in 64 instances (61%). In a post-study survey, the domain experts wrote that CRADLE generally provided a more concise and coherent description of students' activities than did PHATT-E. In all the instances where the domain experts preferred the explanation by PHATT-E, the first action of the log represented the start of a plan a student had abandoned and did not pursue later on. In the explanation set corresponding to these early interaction stages, the values of the average explanation size and frontier size in the given explanation are very low and the evaluators' preferred plan was discarded after the second observation. Inter-annotator agreement between the different experts who judged the same logs was $\kappa = 0.737$ using Fleiss' Kappa measure.

5.2. The TinkerPlots domain

In this section we evaluate CRADLE on another type of open-ended educational system called TinkerPlots, used world-wide to teach students in grades four through eight about statistics and mathematics [Konold and Miller 2004]. Using TinkerPlots, stu-

$$DAE \rightarrow DAE, DCE \quad (5)$$
$$LP = \{1 \prec 2\}$$
$$EC = \{0.s = 1.s, 0.d = 1.d, 0.ei = 1.ei, 0.el = 1.el, 0.s = 2.s, 0.d = 2.d, 0.ei = 2.ei, 0.el = 2.el\}$$
$$MR \rightarrow DAE, DAE, DAE, DAE \quad (6)$$
$$LP = \emptyset$$
$$EC = \{0.s = 1.s, 0.s = 2.s, 0.s = 3.s, 0.s = 4.s, 0.d = 1.d, 0.d = 2.d, 0.d = 3.d, 0.d = 4.d, 1.ei = \text{“}R\text{”}, 2.ei = \text{“}O\text{”}, 3.ei = \text{“}S\text{”}, 4.ei = \text{“}A\text{”}\}$$
$$CSM \rightarrow NS, SAD, MR, SDS, SR, CRT \quad (7)$$
$$LP = \{1 \prec 2, 2 \prec 3, 2 \prec 4, 2 \prec 5, 2 \prec 6\}$$
$$EC = \{0.s = 1.s, 0.s = 2.s, 0.s = 3.s, 0.s = 4.s, 0.s = 5.s, 0.s = 6.s, 2.d = 3.d, 2.d = 6.d\}$$
$$SRP \rightarrow CSM, R, PO \quad (8)$$
$$LP = \{1 \prec 2\}$$
$$EC = \{1.s = 2.s, 2.s = 3.s\}$$

Fig. 11. Exploratory Grammar for TinkerPlots domain

dents build stochastic models and generate pseudo-random samples to analyze the underlying probability distributions. Our study used two different problems for which students interacted with TinkerPlots to model hypothetical situations and to determine the probability of events. There are two key differences between TinkerPlots and VirtualLabs. First, in TinkerPlots, recipes are question dependent and describe ideal solution paths to specific problems. We show later in the section how this affects CRADLE's ability to recognize exogenous actions in students' interactions. Second, although the TinkerPlots grammar is larger than the one used for VirtualLabs and contains significantly more ambiguity, it is not recursive. We will use the following running example problem posed to students using TinkerPlots in schools, called ROSA:

> There are 4 letters printed on cards, each card contains one letter: A,O,R,S. The cards are lined up in a row. After mixing the cards up, what is the probability that the cards would spell ROSA?

In order to solve this problem, the student must perform three subtasks: (1) create a sampler model (the complex action CSM); (2) run the model (R); (3) plot the results (PO). When accomplishing all three subtasks successfully, the student is said to have solved the ROSA problem, which can be represented by the complex action SRP.

This process and the restrictions constraining it are represented using the fourth rule in Figure 11. The third rule shows that in order to achieve the subtask of creating a sampler, the student must perform several actions: create a new sampler (NS), add a device to the sampler (SAD), adjust the device to model the rosa letters (MR), set the number of device spins (SDS), set repetitions (SR) and change replacement type (CRT). In order to achieve a complete “Create a sampler model” (CSM) action, all these sub-actions should be performed. Some of these actions are primitive actions and can be modeled as such, but some can hold additional complexity. For example, as seen in the second rule, when adjusting the device to model the rosa letters (MR),

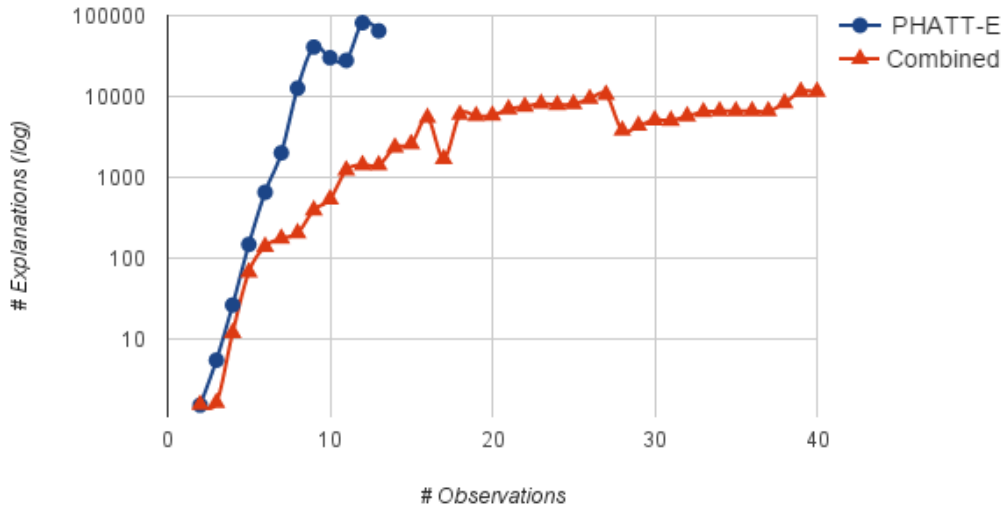


Fig. 12. Number of generated explanations in the TinkerPlots domain

four elements are added to the device (DAE), each one representing one letter of the rosa problem. The student can then decide to rename each of the elements (DCE), but this is optional. Renaming each element is captured using the first rule. The rules described so far are only part of the complete set of rules for this problem.

5.2.1. Speed and termination. We used the CRADLE variant that combined all of the filters described in Section 4.1, which achieved the best runtime performance over using the individual filters. We compared this approach to the PHATT-E algorithm that was introduced in the previous section. The average number of actions in a log is 27.7083 (stdev=14.2721).

The results show that PHATT-E was only able to terminate for two logs out of the 24 logs in the allocated CPU time of 100 seconds. On the same logs, CRADLE managed to generate a solution for 23 logs. The only log for which it failed to produce an output is one of 36 actions, as the algorithm could not finish the run in 100 seconds of CPU time. Where there was a complete plan with no open frontier, CRADLE managed to find it. The average runtime of CRADLE in this domain was 42.9394 seconds (stdev=29.077). These results demonstrate the ability of the CRADLE approach to generalize to new domains.

Figure 12 shows the number of generated explanations for CRADLE and PHATT-E on the different logs. As shown in the figure, CRADLE produces about two orders of magnitude fewer explanations than PHATT-E, the number of explanations being tracked after the twelfth observation.

Plans in the TinkerPlots domain represent solution paths by the student towards solving the ROSA problem. Given that TinkerPlots supports exploratory activities and trial-and-error, we expect CRADLE to be able to distinguish actions representing exogenous actions from those actions representing part of the solution paths. The average number of exogenous actions over the 24 logs we examined was 9.13 (out of 27.7 av-

erage actions per log). This reflects a high degree of exploratory activities by students in the TinkerPlots domain. One example of such an exogenous action occurred when a student set a sampler device to type "urn" and subsequently changed it to a sampler of type "spinner". The grammar requires that creating the sampler device must follow a creation of a sampler mechanism. One of the setting operations will be deemed exogenous by CRADLE given the correct parameter settings.

5.2.2. Agreement with domain expert. We compared the output of the CRADLE algorithm on all of the logs in the TinkerPlots domain to the opinion of a domain expert. We used a single domain expert for this purpose who has significant experience in analyzing students' use of TinkerPlots. The domain expert was presented with a visualization of the most probable explanation generated by CRADLE. Note that in contrast to the VirtualLabs domain, a student's solution in TinkerPlots may be incomplete, and the plan explaining the student's interaction includes predictions about future activities represented by open frontier nodes.

The domain expert was presented with a visualization of the CRADLE generated plan of all 24 logs from the ROSA problem and an additional problem called RAIN (see Apendix), as well as the logs themselves. As in the VirtualLabs domain, the plan was visualized using an interactive GUI that allowed the expert to traverse the trees in the explanation representing the students' solution. Figure 13 shows an example of one of the plans representing a student's solution of the ROSA problem. (Not shown are the parameters of each action, which are displayed in a separate panel.) This solution shows two separate attempts to solve the problem, noted by the two Solve Rosa Problem (SRP) trees. The bottom tree emanating from the SRP action describes an exploratory activity by the student that is an incomplete solution, and the Create Sampler Mechanism (CSM) and PO actions are open frontiers. The top tree emanating from the SRP describes activities that represent the complete solution to the problem.

The domain expert was asked, for each log, whether the explanation produced by CRADLE was correct, in that it provided the best description of the student's activities. For each of the RAIN and ROSA problems, the domain expert confirmed 11 out of 12 explanations generated by CRADLE to be correct (91% accuracy). In both of these erroneous cases, the student used two sampler mechanisms in the solution, but the chosen hypothesis included the wrong one. This result demonstrates the ability of CRADLE to generalize well to new domains despite its inherent incompleteness.

5.3. Comparison with the DOPLAR algorithm

In this section we compare the performance of CRADLE to the DOPLAR algorithm on the synthetic domain used by Kabanza et al. [2013]. DOPLAR is a YAPPR-based plan recognition algorithm, augmented with a weighted model-counting procedure for limiting the number of generated hypotheses at each level. It does so by computing the lower and upper bounds of goal hypothesis probabilities and uses a heuristic that discards explanations that fall below a certain threshold. The algorithm was evaluated on a synthetic domain whose description consists of and-or trees with five possible goals. When translated to rules, it produces 240 actions and approximately 250 rules per problem. Each log in this domain includes nine actions that comprise a plan to satisfy a single goal. Given an observation sequence, DOPLAR outputs a probability distribution over the set of goals, whereas CRADLE outputs a distribution over a set of possible explanations. However, we can compute the probability of each goal in a similar fashion to DOPLAR's approach. We sum each goal's probability for each of the explanations in which it appears, normalizing by total probability of all explanations. Because there is a single goal that was used to create each of the logs, we can compute the mean square error of each algorithm for each instance using the formula

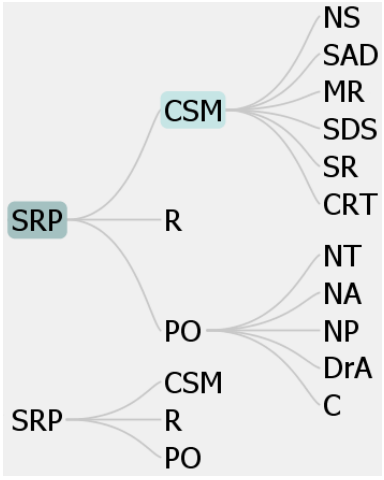


Fig. 13. Visualization of ROSA plan shown to domain expert

$\sum_{g \in G}(P(g) - T(g))^2$ where G is the set of goals, $P(g)$ is the probability assigned to the goal by the algorithm; $T(g)$ is the true probability, which is 1 if g was used to generate the log, and 0 otherwise.

We compared the DOPLAR approach to CRADLE on the same logs generated by Kabanza et al. [2013]. The average error of CRADLE was 0.0439, an order of magnitude smaller than the average error of DOPLAR on this domain, which was 0.3169. In a striking 42 out of 50 cases, CRADLE generated an explanation comprising the true goal, representing a perfect prediction. Effectively, the DOPLAR algorithm uses a single criterion (probability) to prune the hypothesis space, which may explain the lower performance compared to CRADLE which uses an array of filter criteria to prune the hypothesis space.

To illustrate the different behavior of the algorithms, consider the plans in Figures 14 and 15. The first describes the correct plan which can be inferred for the observation sequence $A_{35}, A_{33}, A_{75}, A_{20}, A_{25}, A_{75}, A_{63}, A_{99}, A_{48}$. The second describes a partial plan that can be generated for the first observation. The second observation, however, cannot be combined in any way in the plan with B_{112} in its root. DOPLAR keeps a record of every possible goal that can match every observation, even if it cannot be developed further. Consequently, DOPLAR gave a probability confidence window in the range of 0 to 0.0938 to the goal B_{112} that is pursued in this plan. In CRADLE, on the other hand, the partial plan shown in Figure 15 will be omitted when considering the second observation, since it cannot be combined into the plan. This is an especially simple scenario, but even more subtle scenarios favor CRADLE. Consider a case in which the second observation could have been introduced into a separate tree in the explanation. Even here, the CRADLE algorithm would have preferred to eliminate this explanation, since it contains two plans when there is a better plan (the one from Figure 14) which contains only one. Accordingly, CRADLE gave a probability of 1 for the pursued goal to be B_{140}, managing to find the correct explanation and matching it perfectly.

6. CONCLUSION

In exploratory domains, agents' behavior is characterized by interleaving of activities, exogenous actions, and mistakes. We provided a new heuristic plan recognition algorithm for exploratory domains that leads to significant improvement as compared

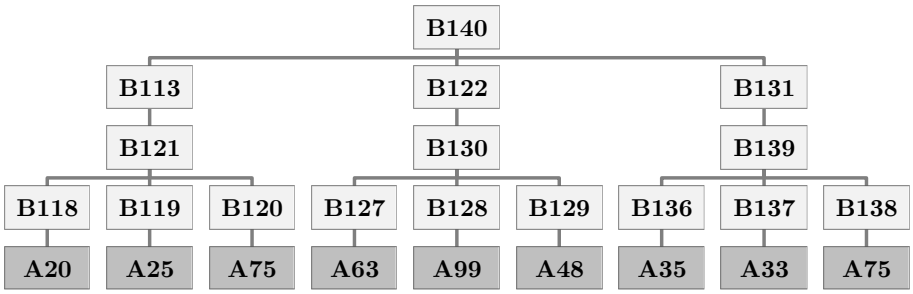


Fig. 14. Correct plan for DOPLAR scenario

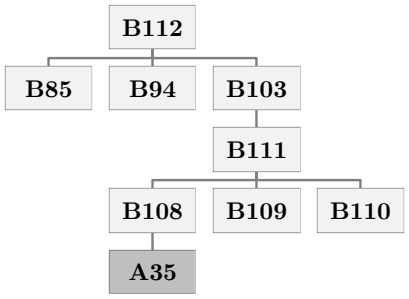


Fig. 15. Incorrect partial plan for DOPLAR scenario

to the state of the art when evaluated on real data. CRADLE extends existing plan recognition approaches by using domain-independent heuristics to prune the space of possible explanations, by explicitly reasoning about exogenous actions and mistakes, and by updating plan arguments so that explanations remain consistent with new observations. CRADLE was able to produce better explanations than two previously proposed algorithms, PHATT [Geib and Goldman 2009] and DOPLAR [Kabanza et al. 2013] when evaluated on real-world data sets in which agents engage in exploratory behavior.

We are currently pursuing work with CRADLE in several directions. First, we are evaluating CRADLE on a large-scale intrusion detection domain. Second, we are using CRADLE to provide machine-generated support to users as well as to construct interactive visualizations of users' activities to overseers (for example, teachers or system administrators). Third, we will devise filters for reasoning about exploration early on in the agent's interaction. Another direction for future research would be to apply similar capabilities as CRADLE's to additional plan recognition algorithms, such as ELEXIR, a plan recognition method based on combinatory categorial grammars [Geib 2009].

7. ACKNOWLEDGMENTS

This work was supported in part by EU FP7 FET project, Grant agreement n.600854, and the Israeli Science Foundation Research Grant no. 1276/12.

A. QUESTIONS USED

We detail the questions used in evaluating CRADLE on the two educational software domains.

VirtualLabs questions

ORACLE. Given four substances A, B, C, and D that react in a way that is unknown, design and perform virtual lab experiments to determine which of these substances react, including their stoichiometric coefficients.

UNKNOWN ACID. The cabinet contains a solution labeled "Unknown Acid," which is a weak mono-protic acid with an unknown Ka and with an unknown concenration. Your job is to determine the concentration and Ka to two significant figures. Please report your results and explain your procedure.

TinkerPlots questions

ROSA. Jessica has 4 letters printed on cards: R, O, S, and A. After mixing them up, she blindly picks the 4 letters one at a time and arranges them in line in the order she chose them. Build a TinkerPlots model and use it to help you estimate the probability of Jessica spelling the word ROSA.

RAIN. There is a 75% chance of rain for each of the next 4 days. Build a TinkerPlots model and use it to help you estimate the probability of getting rain on all 4 days.

REFERENCES

O. Amir and Y. Gal. 2013. Plan recognition and visualization in exploratory learning environments. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 3, 3 (2013), 16.

D. Avrahami-Zilberbrand and G.A. Kaminka. 2005. Fast and complete symbolic plan recognition. In *Proc. 22nd International Joint Conference on Artificial Intelligence (IJCAI'05)*. 653–658.

D. Avrahami-Zilberbrand and G.A. Kaminka. 2007. Incorporating observer biases in keyhole plan recognition (efficiently!). In *Proc. 22nd Conference on Artificial Intelligence (AAAI'07)*, Vol. 7. 944–949.

N. Blaylock and J.F. Allen. 2004. Statistical goal parameter recognition. In *Proc. 14th International Conference on Automated Planning & Scheduling (ICAPS'04)*, Vol. 4. 297–304.

N. Blaylock and J. F. Allen. 2006. Hierarchical instantiated goal recognition. In *Proc. 21st Conference on Artificial Intelligence (AAAI'06) workshop on modeling others from observations*. 8–15.

H.H. Bui. 2003. A general model for online probabilistic plan recognition. In *Proc. 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, Vol. 3. 1309–1315.

T.V. Duong, H.H. Bui, D.Q. Phung, and S. Venkatesh. 2005. Activity recognition and abnormality detection with the switching hidden semi-markov model. In *Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Vol. 1. IEEE, 838–845.

M. S. Fagundes, F. Meneguzzi, R. H. Bordini, and R. Vieira. 2014. Dealing with ambiguity in plan recognition under time constraints. In *Proc. 13th international conference on autonomous agents and multi-agent systems (AAMAS'14)*. International Foundation for Autonomous Agents and Multiagent Systems, 389–396.

Y. Gal, S. Reddy, S. Shieber, A. Rubin, and B. Grosz. 2012. Plan recognition in exploratory domains. *Artificial Intelligence* 176, 1 (2012), 2270 – 2290.

Y. Gal, O. Uzan, R. Belford, M. Karabinos, and D. Yaron. 2015. Making sense of students' actions in an open-ended virtual laboratory environment. *Journal of Chemical Education* 92, 4 (2015), 610–616.

G. Gazdar and G.K. Pullum. 1981. Subcategorization, constituent order, and the notion head. In *The Scope of Lexical Rules*, M. Moortgat, H. v.d. Hulst, and T. Hoekstra (Eds.). Foris, Dordrecht, Holland, 107–123.

C.W. Geib. 2009. Delaying commitment in plan recognition using combinatory categorial grammars. In *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*. 1702–1707.

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

C.W. Geib, J. Maraist, and R.P. Goldman. 2008. A new probabilistic plan recognition algorithm based on string rewriting. In *Proc. 18th International Conference on Automated Planning and Scheduling (ICAPS'08)*. 91–98.

F. Kabanza, J. Filion, A.R. Benaskeur, and H. Irandoust. 2013. Controlling the hypothesis space in probabilistic plan recognition. In *Proc. 23rd International Conference on Artificial Intelligence (IJCAI'13)*. 2306–2312.

S. Katz, J. Connelly, and C. Wilson. 2007. Out of the lab and into the classroom: An evaluation of reflective dialogue in ANDES. *Frontiers in Artificial Intelligence and Applications* 158 (2007), 425.

C. Konold and C. Miller. 2004. *TinkerPlots dynamic data exploration 1.0*. Key Curriculum Press. http://www.keypress.com/x5715.xml

D.V. Pynadath and M.P. Wellman. 2000. Probabilistic state-dependent grammars for plan recognition. In *Proc. 16th Conference on Uncertainty in Artificial Intelligence (UAI'00)*. Morgan Kaufmann Publishers Inc., 507–514.

G. Sukthankar and K.P. Sycara. 2008. Hypothesis pruning and ranking for large plan recognition problems. In *Proc. 23rd AAAI Conference on Artificial Intelligence (AAAI'08)*, Vol. 8. 998–1003.

O. Uzan, R. Dekel, O. Seri, and Y. Gal. 2015. Plan recognition for exploratory learning environments using interleaved temporal search. *AI Magazine* 36, 2 (2015), 10–21.

S. Wiseman and S. Shieber. 2014. Discriminatively reranking abductive proofs for plan recognition. In *Proc. 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*. AAAI Publications, 380–384.

D. Yaron, M. Karabinos, D. Lange, J.G. Greeno, and G. Leinhardt. 2010. The ChemCollective–virtual labs for introductory chemistry courses. *Science* 328, 5978 (2010), 584.