Goal and Plan Recognition Design for Plan Libraries

REUTH MIRSKY, KOBI GAL, RONI STERN, and MEIR KALECH,
Ben-Gurion University of the Negev

This article provides new techniques for optimizing domain design for goal and plan recognition using plan libraries. We define two new problems: Goal Recognition Design for Plan Libraries (GRD-PL) and Plan Recognition Design (PRD). Solving the GRD-PL helps to infer *which* goal the agent is trying to achieve, while solving PRD can help to infer *how* the agent is going to achieve its goal. For each problem, we define a worst-case distinctiveness measure that is an upper bound on the number of observations that are necessary to unambiguously recognize the agent's goal or plan. This article studies the relationship between these measures, showing that the worst-case distinctiveness of GRD-PL is a lower bound of the worst-case plan distinctiveness of PRD and that they are equal under certain conditions. We provide two complete algorithms for minimizing the worst-case distinctiveness of plan libraries without reducing the agent's ability to complete its goals: One is a brute-force search over all possible plans and one is a constraint-based search that identifies plans that are most difficult to distinguish in the domain. These algorithms are evaluated in three hierarchical plan recognition settings from the literature. We were able to reduce the worst-case distinctiveness of the domains using our approach, in some cases reaching 100% improvement within a predesignated time window. Our iterative algorithm outperforms the brute-force approach by an order of magnitude in terms of runtime.

CCS Concepts: • **Computing methodologies → Planning for deterministic actions**;

Additional Key Words and Phrases: Plan recognition, plan libraries, environment design, domain design, Conflict Based Search

ACM Reference format:
Reuth Mirsky, Kobi Gal, Roni Stern, and Meir Kalech. 2019. Goal and Plan Recognition Design for Plan Libraries. *ACM Trans. Intell. Syst. Technol.* 10, 2, Article 14 (January 2019), 23 pages.
https://doi.org/10.1145/3234464

1 INTRODUCTION

Plan and goal recognition, the tasks of inferring an agent's plans and goals from a sequence of observations, are cornerstone problems in Artificial Intelligence [42]. By giving a recognizer agent the ability to reason about the goals and future actions of other actors, it can make more informed choices for its own plans and improve collaboration or reduce interruptions [12].

Recently, there has been a surge of interest in applying plan and goal recognition algorithms to real-world problems, such as e-learning [3, 13, 47], medical informatics [2, 9, 18, 36], and cyber security application [29, 31]. Such settings require us to be able to recognize, in real time, agents' plans and goals. To name a few examples: In e-learning settings, identifying a student whose activities in educational software deviates substantially from that of other students can be important

Authors' addresses: R. Mirsky, K. Gal, R. Stern, and M. Kalech, Software and Information Systems Engineering Department, Building 96, Marcus Campus, Beer Sheva, 84105; emails: {dekelr, kobig, sternron, kalech}@bgu.ac.il.

2157-6904/2019/01-ART14 $15.00
https://doi.org/10.1145/3234464

Table 1. The Relationship between the GRD-PL and PRD Problems

Problem	Input	Output	Metric	Meaning of Metric
GRD	STRIPS domain	STRIPS domain with less actions	*wcd*	Maximum number of observations required to unambiguously infer the **goal** of the agent
GRD-PL	Plan Library	Plan Library with less rules	*wcd*	Maximum number of observations required to unambiguously infer the **goal** of the agent
PRD	Plan Library	Plan Library with less rules	*wcpd*	Maximum number of observations required to unambiguously infer the **plan** of the agent

to notify a teacher or to guide a student [37]; recognizing the treatment procedure a physician has decided to use can be useful to provide alerts when there are deviations from the treatment, or to suggest alternative treatments; recognizing how a hacker plans to attack a system in real-time can assist in defending the system.

Plan libraries model agents' activities using hierarchies of basic and complex actions [8, 19, 46]. They offer a complimentary approach to generative (STRIPS-like) representations of agents activities [32, 33, 40, 43]. The advantage of using plan libraries is threefold: First, plans capture the inherent hierarchical structure that is endemic to many task settings in the real world (e.g., military operations, student learning, advice provision, and more). In particular, they are able to capture complex agent behavior, such as interleaving and exploratory behavior [28]. Second, plans can encode prior information about how agents perform tasks in the domain. Third, they can be presented to people in a way that facilitates their understanding of a hierarchical task decomposition [38]. Due to these advantages, plan recognition using plan libraries is an active area of research in AI [5, 6, 17, 26].

The focus of this article is on methods for optimizing the design of plan libraries to facilitate plan recognition. It defines a measure of worst-case distinctiveness (*wcd*) to describe the maximum number of observations needed to recognize an agent's goal. The worst-case distinctiveness (*wcd*) was first proposed by Keren et al. [20] for domains based on STRIPS representation [11].[1] It provides methods for reducing "recognition ambiguity" in a domain, which occurs when there are several possible plans that can describe a given observation sequence, though only one plan is correct [10, 27].

The first contribution of this work is to adapt the *wcd* measure of Keren et al. to settings represented by hierarchical plan libraries. The second contribution of this work is to extend the notion of worst-case distinctiveness from the context of goal recognition design to plan recognition design. This worst-case distinctiveness for plans (*wcpd*) measures the number of observations needed to unambiguously identify the agent's plan.

We define two types of problems: Goal recognition design for plan libraries (GRD-PL) and plan recognition design (PRD). Both problems deal with optimizing the design of plan libraries to facilitate recognition of an agent's goals or plans, respectively. Solving a GRD-PL problem will yield a domain design in which it is easier (i.e., require fewer observations) to explain *which* goal the agent is trying to achieve. Solving a PRD problem will yield a domain design in which it is easier to explain *how* the agent is going to achieve its goal. Table 1 summarizes the differences between the two problems and the original GRD problem.

[1]STRIPS is a state-based world model based on first-order predicate calculus formulas. A classical plan for a STRIPS-based planning problem will be a sequence of actions (operators) from an initial state to a goal state.

As a running example, consider an on-line banking company that allows users to perform various actions such as cash withdrawal, money transfer, update account, and so on. A customer wishing to transfer funds must pass an identification process (by login or pin code), choose which account to access (savings or checking), choose the amount to withdraw, and approve the transaction. Each of these steps can be modeled as a complex level action that can be carried out in various ways, such as using a mobile device or personal computer, by phone, or with the assistance of a clerk. Each such complex level action can be decomposed into a series of basic-level actions.

Now suppose the system has observed a client logging in from a new device in a foreign country and requesting to withdraw funds. It is important for the system to infer whether the login represents one of several types of fraudulent activities. For example, withdrawing funds beyond the limit of the customer, and stealing funds by transferring to a different account. Reducing the worst-case distinctiveness measure for this domain will decrease the number of actions that the system needs to observe before being able to unambiguously distinguish between these two activities.

The third contribution of this article is to provide two algorithms for domain design using plan libraries that optimize the worst-case distinctiveness measures of the domain while still allowing the agent to fulfill its goals. The first is a brute-force (BF) algorithm that searches over all design options of the domain. The second algorithm, ReduceCBS, is a constraint-based search approach that iteratively identifies the critical plans in the domain that are most difficult to distinguish and removes them.

The fourth contribution of this article is the evaluation of the GRD-PL and PRD paradigms on three known settings from the literature. We compute the value of the worst-case distinctiveness in these settings and its computational load in terms of runtime. We evaluate the performance of both algorithms for reducing the *wcd* and *wcpd* values, both in terms of possible reduction and in terms of runtime. We show that it is possible to reduce the worst-case distinctiveness of the domain using our approach, in some cases reaching 100% improvement. Moreover, the ReduceCBS approach outperforms the brute-force approach both in terms of *wcd* and *wcpd* improvement within a predesignated time window, and in terms of runtime by an order of magnitude.

Last, we discuss the tradeoffs between using the *wcd* or *wcpd* as tools for reducing rules in the plan library, in the three different settings. This work has implications for domain designers. First, in providing a way to measure the inherent ambiguity in a plan library using the worst-case distinctiveness measures, and then to reduce the ambiguity using the new algorithms for plan library design.

The rest of this article is organized as follows: Section 2 formally defines plan libraries and their plans; Section 3 formally defines the *wcd* and *wcpd* metrics in the context of plan libraries, introduces the GRD-PL and PRD problems, and discusses the relationship between *wcd* and *wcpd*; Section 4 presents the algorithmic contributions: computation of the *wcd* and *wcpd* metrics and two optimization techniques for reducing the plan libraries to minimize these metrics; Section 5 presents the complexity analysis for computing the measures and for the optimization algorithms; Section 6 shows the empirical results of this work on both reduction techniques and on the relationship between the *wcd* and *wcpd* metrics; Section 7 discusses related work; Section 8 concludes this work and proposes future work.

2 DEFINITIONS

We use the standard definition of a plan library from the plan recognition literature [16, 18, 27].

Definition 1. A plan library is a tuple $L = \langle B, C, G, R \rangle$, where B is a finite set of basic actions, C is a finite set of complex actions, $G \subseteq C$ the possible goals and R is a set of rules of the form

$c \rightarrow \tau \mid \prec_\tau$, where $c \in C$, τ is a string from $(B \cup C)^*$ and $\prec_\tau = \{(i,j) \mid c_i \prec c_j\}$ where c_i, c_j refer to the ith and jth actions in τ, respectively.

The set B represents all of the observable actions an agent can execute, C represents more complex or abstract actions that should be taken to achieve the goals G. Each $r \in R$ represents how a complex action from C can decompose to a sequence of basic and complex actions. For $c_i, c_j \in \tau$, we say that $c_i \prec c_j$ if there exists an ordering constraint $(i < j) \in \prec_\tau$. Each basic action b has a replica in the set of complex actions c_b and a rule $c_b \rightarrow b$. As shown in previous work by Geib and Goldman [16], these rules can be artificially added to the plan library and they are required to isolate exactly one action by removing exactly one rule from the plan library.

A partial plan library for our running example is shown below.

Set of basic actions $B = \{$identification, useCard, withdrawal, transfer$\}$.

Set of complex actions $C = \{\underline{\text{CashWithdrawal(CW)}}, \underline{\text{MoneyTransfer(MT)}}, \underline{\text{Identification}}, \underline{\text{UseCard}}, \underline{\text{Withdrawal}}, \underline{\text{Transfer}}\}$.

Set of rules $R =$

$$\begin{aligned}
\{\underline{\text{CW}} &\rightarrow (\underline{\text{Identification}}, \underline{\text{Withdrawal}}) \mid \prec_\tau = \{(1,2)\},\\
\underline{\text{MT}} &\rightarrow (\underline{\text{Identification}}, \underline{\text{Transfer}}) \mid \phi,\\
\underline{\text{MT}} &\rightarrow (\underline{\text{UseCard}}, \underline{\text{Transfer}}) \mid \prec_\tau = \{(1,2)\}\\
\underline{\text{Identification}} &\rightarrow (\text{identification}) \mid \phi\\
\underline{\text{Withdrawal}} &\rightarrow (\text{withdrawal}) \mid \phi\\
\underline{\text{Transfer}} &\rightarrow (\text{transfer}) \mid \phi\\
\underline{\text{UseCard}} &\rightarrow (\text{useCard}) \mid \phi\}
\end{aligned}$$

An agent can either withdraw cash or transfer money. The basic actions that can be observed are identification, withdrawal, transfer, and useCard. Note that there is a single rule for executing a $\underline{\text{CW}}$ plan, which requires user identification to precede transfer. There are two different ways to perform a money transfer—one is by identification and transfer (in either order) and the other is by using a credit card and then performing the transfer.

Definition 2. A plan is a (partially) ordered tree $p = (V, E, \mathcal{L})$, where V are vertices, E are edges, and $\mathcal{L}$ is a labeling function $\mathcal{L} : V \rightarrow B \cup C$ mapping every node in the tree to either a basic or a complex action in the plan library. For every node c and its children τ there exists a rule in the plan library $c \rightarrow \tau \mid \prec_\tau$, such that the order over the children nodes does not violate $\prec_\tau$.

The root node of a plan is referred to as its goal, and is labeled with a complex action from G. Each inner node is labeled with a complex action such that its children nodes are a decomposition of its complex action into constituent actions according to one of the rules. The set of all leaves of a plan p is denoted by $leaves(p)$, which are a subset of the actions from $B \cup C$. The notation $c_i \prec c_j$ is used to denote a pair of leaves c_i and c_j of a plan p where c_i must be performed before c_j. This occurs if there is such an ordering constraint in the rules.

Given a plan library PL as defined in Definition 1 and a goal $g_i \in G$, we define $Plans(g_i)$ to be the set of all plans with the root node labeled as g_i. Given a plan p_i, we define $Goal(p_i)$ to be the label of the root of p_i.

Figure 1 shows two of the plans in our banking example. Plan p_1 is a plan that represents how the $\underline{\text{CW}}$ goal can be achieved, by performing two actions of identification, withdrawal. The dashed arrow represents the ordering constraint between the actions. Plan p_2 represents one way of how a $\underline{\text{MT}}$ goal can be achieved, by performing two actions of identification, transfer.

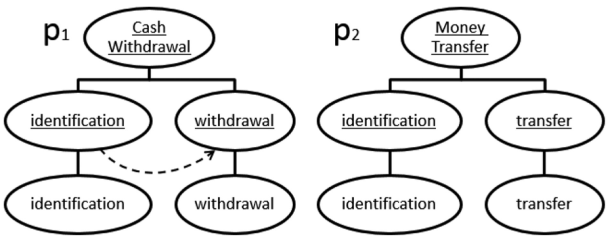

Fig. 1. Two plans for withdrawing cash and transferring money.

An *observation sequence* is an ordered set of basic actions that represents actions carried out by the observed agent. The following definition states that a plan *describes* an observation sequence O iff every observation is mapped to a leaf in the tree in an order that is consistent with the ordering constraints of the plan.

Definition 3. A plan $p = (V, E, \mathcal{L})$ describes an observation sequence O iff there exists a partial function $f : O \rightarrow leaves(p) \cap B$ such that $\forall o \in O \, \exists v \in V \, f(o) = v$, and $\forall o_i, o_j \in O \; i < j \rightarrow \neg(f(o_j) \prec f(o_i))$.

The last condition is needed to enforce the ordering constraints. In particular, if p describes a sequence of observations $O = \{o_1, \ldots, o_n\}$ it also describes all prefixes of O, i.e., observation sequences $\{o_1, \ldots, o_l\}, l \leq n$. Figure 1 shows two plans based on the example plan library. Plan p_1 describes the following observation sequences: ⟨identification, withdrawal⟩ and ⟨identification⟩.

For each plan p, we define $OBS(p)$ as the set of all observation sequences that are described by p. This definition can be extended to a set of plans P, such that $OBS(P)$ is the set of all observation sequences that are described by some plan in P. Formally, $OBS(P) = \bigcup_{p \in P} OBS(p)$. In our example plan library, we get that $OBS(p_2) =$ {⟨identification, transfer⟩, ⟨transfer, identification⟩ ⟨identification⟩, ⟨transfer⟩}, since in a money transfer plan, providing the transfer details and identification action can be performed in any order.

3 THE GRD-PL AND PRD PROBLEMS

In this section, we present the worst-case distinctiveness measure for plan libraries and define the Goal Recognition Design for Plan Libraries (GRD-PL). Then, we present the worst-case *plan* distinctiveness measure, discuss its relation with the worst-case distinctiveness measure, and define the Plan Recognition Design (PRD) problem.

Before we elaborate on these definitions, we detail the assumptions we make in this work:

- Determinism—actions are deterministic, in the sense that an observation is mapped to exactly one basic action in the plan library.
- Observability—the agent and the system are fully observable. That is, no action that the agent performs is hidden, and the plan library is known.

Keren et al. [20] defined the *wcd* measure in the context of a STRIPS-based domain description $P_D = \langle F, I, A \rangle$, where F are the fluents, $I \subseteq F$ is an initial state and A are the actions. Specifically, the *wcd* of any two goals g, g' is the longest sequence of observations required to observe before there is no ambiguity about the goals that the acting agent is pursuing. Using the *wcd* measure, they defined the Goal Recognition Design problem:

Definition 4. The Goal Recognition Design (GRD) problem is defined as a tuple $D = \langle P_D, G_D \rangle$ $\forall g \in G_D, g \subseteq F$ is a goal. The output of a solver for the GRD problem is $P'_D = \langle F, I, A' \rangle$ such that $A' \subseteq A$ and it holds that $wcd(P'_D) \leq wcd(P''_D)$ without restricting the agent from achieving any of the goals in G_D.

3.1 Goal Recognition Design for Plan Libraries

The Goal-Recognition Design Problem with Plan Libraries (GRD-PL) extends the GRD paradigm of Keren et al. [20] to plan libraries. Their work and representation is STRIPS-based, and thus plans are represented as a sequence of actions, rather than hierarchies of basic and complex actions as in plan libraries. Two key steps are needed to extend this problem: (1) Describing the notion of distinctiveness in hierarchical plans; and (2) Defining the *wcd* measure for hierarchical plans. To this end, we first define the notion of distinctiveness between two plans.

Given a plan library and two plans p_1, p_2, the two plans are *distinct* if it is possible to unambiguously recognize them from the observations that describe the plans. Formally, we say that p_1 and p_2 are distinct iff $OBS(p_1) \cap OBS(p_2) = \emptyset$. We can now define the worst-case distinctiveness measure between two goals as follows:

Definition 5. Let $P_1 = Plans(g_1), P_2 = Plans(g_2)$ for $g_1, g_2 \in G$. The *wcd* of g_1, g_2 is the longest observation sequence that is described by both a plan in P_1 and a plan in P_2:

$$wcd(g_1, g_2) = \max_{O \in OBS(P_1) \cap OBS(P_2)} |O|. \tag{1}$$

If $g_1 = g_2$, then we define $wcd(g_1, g_2) = 0$.

We extend the *wcd* definition to a plan library P_D and say that $wcd(P_D) = \max_{g_i, g_j \in G} wcd(g_i, g_j)$, where G is the set of goals in P_D. In our example plan library, the *wcd* of the plan library is 1. To see this, suppose that the first observation is identification. In this case both plans p_1 and p_2 are still possible, so identification cannot be used to distinguish between the goals $\underline{\text{MT}}$ and $\underline{\text{CW}}$. However, any observation sequence of size 2 or more can be used to distinguish the goal of the agent. We therefore have $wcd(\underline{\text{MT}}, \underline{\text{CW}}) = 1$ and similarly *wcd* of the example plan library is 1. Using these definitions, we can define the Goal Recognition Design Problem for Plan Libraries (GRD-PL) as follows:

Definition 6 (wcd Reduction). A *wcd-reduction* of a plan library $P_D = \langle B, C, G, R \rangle$ is a plan library $P'_D = \langle B, C, G, R' \rangle$ such that $R' \subset R$, $wcd(P'_D) < wcd(P_D)$, and every goal in G is achievable in P'_D. That is, for every goal $g \in G$, $Plans(g) \neq \emptyset$.

For the rest of the article, we will refer to *wcd* reduction simply as reduction.

Definition 7. A Goal Recognition Design Problem for Plan Libraries (GRD-PL) is defined by a plan library P_D. A solution to a GRD-PL problem is a reduction P'_D for P_D, and an optimal solution is a reduction P'_D such that there is no other reduction P''_D for which $wcd(P'_D) \leq wcd(P'_D)$.

3.2 Plan Recognition Design

Plan Recognition Design (PRD) is the problem of designing a domain in a way that will allow faster recognition of the plan of an acting agent. While in GRD-PL the design reduces the number of observations required to unambiguously recognize which *goal* the agent is pursuing, PRD reduces the number of observations required to unambiguously recognize which *plan* the acting agent is using.

For this purpose, we need a new metric, *worst-case plan distinctiveness (wcpd)*, which is defined as the number of observations needed in the worst case to recognize the agent's plan.

Definition 8. Given two plans p_1, p_2, we define the worst-case plan distinctiveness (wcpd) to be

$$wcpd(p_1, p_2) = \max_{O \in OBS(p_1) \cap OBS(p_2)} |O|. \tag{2}$$

If $p_1 = p_2$, then we define $wcpd(p_1, p_2) = 0$.

We extend this definition to plan libraries, such that the *wcpd* of a plan library P_D is the maximum *wcpd* of every pair of plans that can be generated for goals in P_D. Formally, we have that

$$wcpd(P_D) = \max_{p_1, p_2 \in \bigcup_{g \in G} Plans(g)} wcpd(p_1, p_2). \tag{3}$$

The *wcpd* of the plan library from our running example is 1, since after observing identification there is ambiguity regarding the plan of the agent (whether to withdraw money or to perform a transfer). Note that the *wcpd* and *wcd* of the example are both equal to 1, this need not be the case in general. We show this in the next section.

Similar to Definition 6, we define a plan library P'_D to be a *wcpd-reduction* of a plan library P_D if P'_D has a subset of the rules in P_D, it has a smaller *wcpd* and every goal in G is achievable in P'_D. We are now ready to define the new problem of Plan Recognition Design, which aims to minimize the plan library's *wcpd*.

Definition 9. The problem of Plan Recognition Design (PRD) is defined as a tuple $D = \langle P_D, L_D \rangle$, where P_D is a domain description represented by a plan library $\langle B, C, G, R \rangle$ and L_D is a set of possible plans such that $L_D = \bigcup_{g \in G} Plans(g)$. The output of a PRD problem is a *wcpd* reduction of P_D. An optimal solution to a PRD problem is a *wcpd* reduction P'_D such that there is no other *wcpd* reduction having a smaller *wcpd*.

3.3 The Relationship Between *wcd* and *wcpd*

In this section, we discuss the relationship between the two measures *wcd* and *wcpd*. We show that (1) the *wcd* of two goals is equal to the maximum *wcpd* of the plans for achieving those goals (Lemma 1); (2) The *wcpd* of the entire plan library is at least as high as the *wcd*. The empirical implications of these results are shown in Section 6.

Lemma 1. *The wcd of two goals is equal to the maximum wcpd of the plans for achieving these goals:*

$$wcd(g_i, g_j) = \max_{\substack{p_i \in Plans(g_i), \\ p_j \in Plans(g_j).}} wcpd(p_i, p_j) \tag{4}$$

Proof. Let g_1, g_2 be two goals and $P_i = Plans(g_i)$ for $i = 1, 2$. Suppose that $wcd(g_1, g_2) = m$. Thus m is the length of the longest observation sequence in $OBS(P_1) \cap OBS(P_2)$. According to Definition 5, there must be two plans $p_1 \in P_1, p_2 \in P_2$ with respective observation sequences $O^* \in OBS(p_1) \cap OBS(p_2)$ such that $\mid O^* \mid = m$. Thus, m is the length of the longest observation sequence in $OBS(P_1) \cap OBS(P_2)$. According to Definition 8, this means that $wcpd(p_1, p_2) = \max_{O \in OBS(p_1) \cap OBS(p_2)} |O| = m$.

Since p_1, p_2 are in P_1, P_2, respectively, we get that

$$\max_{p_i \in P_1, p_j \in P_2} wcpd(p_i, p_j) \geq wcpd(p_1, p_2) = m. \tag{5}$$

Assume by contradiction that there are plans $p'_1 \in P_1, p'_2 \in P_2$ such that $wcpd(p'_1, p'_2) > wcpd(p_1, p_2)$. In such a case, $wcd(g_1, g_2) = wcd(Goal(p'_1), Goal(p'_2))$, but this contradicts $wcd(g_1, g_2) = m$.

Thus, $wcd(g_1, g_2) \leq \max_{p_1 \in Plans(g_1), p_2 \in Plans(g_2)} wcpd(p_i, p_j)$. □

To illustrate, Figure 2 shows four different plans of a plan library P_D, where we assume that the actions must be fully ordered. $\underline{\text{CW}}$ and $\underline{\text{MT}}$ are two goals from P_D, $Plans(\underline{\text{CW}}) = \{p_1, p_2\}$ and $Plans(\underline{\text{MT}}) = \{p_3, p_4\}$. We know that the *wcd* of $\underline{\text{CW}}$ and $\underline{\text{MT}}$ is 2, since observing $\langle \text{identification}, \text{changeSum} \rangle$ does not distinguish between p_2 and in p_4. This is also the longest

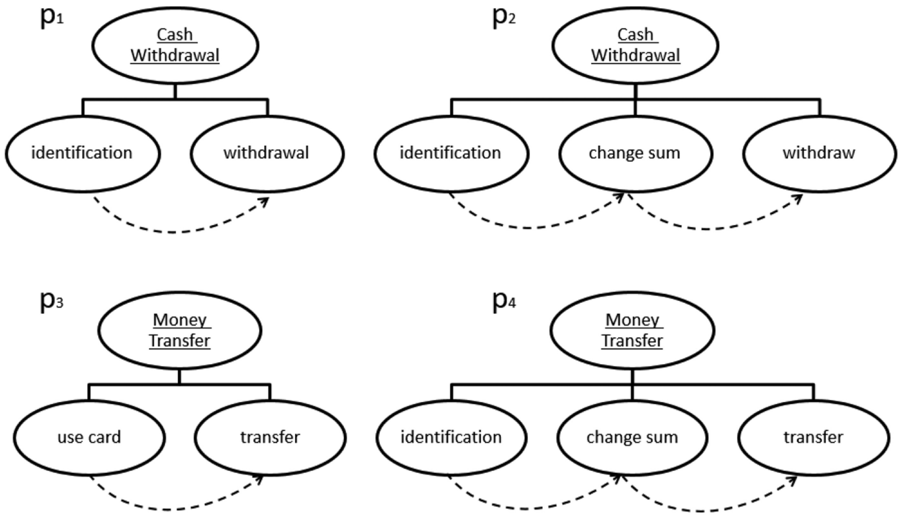

Fig. 2. Plans for CashWithdrawal and MoneyTransfer actions.

observation sequence that is required to distinguish between any two plans. Therefore, we have that $wcpd(p_2, p_4) = 2$.

While in this example, the *wcpd* of two plans equals the *wcd* of their respective goals, in some cases the *wcpd* of plans may be strictly smaller than the *wcd* of their goals. For instance, in the example from Figure 2, $wcd(Goal(p_1), Goal(p_3)) = wcpd(p_2, p_4) = 2$ but $wcpd(p_1, p_3) = 0$.

However, the following shows that the *wcpd* of a complete plan library cannot be smaller than its *wcd*.

Theorem 1. *For every plan library P_D it holds that $wcd(P_D) \leq wcpd(P_D)$.*

Proof. Let $G_D = L_D \times L_D$ be the set of all possible pair of plans in L_D. We divide G_D into two groups: $G_= = \{\langle p_i, p_j\rangle \in G_D s.t. Goal(p_i) = Goal(p_j)\}$, and $G_{\neq} = \{\langle p_i, p_j\rangle \in G_D s.t. Goal(p_i) \neq Goal(p_j)\}$. $G_=$ is the set of all pair of plans with the same goal, and $G_{\neq}$ is the set of all pair of plans with different goals. By Equation 3, $wcpd(P_D) = \max_{\langle p_1,p_2\rangle \in G_D} wcpd(p_1, p_2)$. Now consider any two plans $\langle p_1, p_2\rangle = \text{argmax}_{\langle p_1,p_2\rangle \in G_D} wcpd(p_1, p_2)$. By definition, $wcdp(P_D) = wcdp(p_1, p_2)$. There are two possibilities. If $\langle p_1, p_2\rangle \in G_=$, then we know that $Goal(p_1) = Goal(p_2)$. We get that

$$wcpd(P_D) = wcpd(p_1, p_2) \geq wcd(Goal(p_1), Goal(p_2)) = wcd(g, g) = 0, \tag{6}$$

because $wcd(g, g)$ is defined to be 0. We get that if $\langle p_1, p_2\rangle \in G_{\neq}$, then we know that $Goal(p_1) \neq Goal(p_2)$, and then

$$wcpd(p_1, p_2) = \max_{\substack{p_i \in Plans(Goal(p_1)),\\ p_j \in Plans(Goal(p_2))}} wcpd(p_i, p_j). \tag{7}$$

Lemma 1 shows that the maximum *wcpd* of two plans for different goals, equals to the *wcd* of these goals. Hence,

$$\max_{\substack{p_i \in Plans(Goal(p_1)),\\ p_j \in Plans(Goal(p_2))}} wcpd(p_i, p_j) = wcd(Goal1(p_1), Goal(p_2)) = wcd(G_{\neq}). \tag{8}$$

□

The value of the *wcpd* is strictly greater than that of *wcd* when there exist two plans with the same goal that describe an observation sequence that is longer than the *wcd*. This commonly

ALGORITHM 1: $wcd(P_D)$

```
Input: $P_D$: the plan library.
Output: $wcd$: the length of the longest non-distinct sequence in $P_D$.
1 $i \leftarrow 0$
2 $P_0 \leftarrow$ InitialPlans($G$)
3 $isDistinct \leftarrow False$
4 while $!isDistinct$ do
5     $i++$
6     $P_i \leftarrow$ AddOneObs($P_i$)
7     $isDistinct \leftarrow$ Distinct($P_i, G$)
8 return $i - 1$
```

occurred in our empirical evaluation. To illustrate, consider the plans from Figure 2, excluding p_4. In such a set of plans, the *wcd* of the plan library is 0, since $wcd(\underline{\text{CW}}, \underline{\text{MT}}) = 0$ (the first observation unambiguously recognizes the goal of the agent). However, for the pair of plans p_1, p_2 it holds $wcpd(p_1, p_2) = 1$, because observing identification does not reveal which plan is used by the agent to fulfill CW. The *wcpd* of the plan library including p_1, p_2, p_3 is 1, strictly larger than its *wcd*.

4 SOLVING THE GRD-PL AND PRD PROBLEMS

In this section, we show that the *wcd* and *wcpd* measures can be used to optimize the design of plan libraries to aid plan recognition. The GRD approach [20] reduces the *wcd* of a domain in a STRIPS space by removing actions from the model without hindering the ability of the agent to reach its goals. Analogously to the GRD formalization, we show that it is possible to reduce the *wcd* and the *wcpd* of a given plan library by removing rules from it, in a way that does not hinder the agents ability to reach all goals. To illustrate the analogy between the GRD approach and our own, let us extend the running example to include a robber, and suppose that R1 and R2 are two robbery scenarios, but R1 starts with approaching a teller with a hidden gun and R2 starts with running into the bank with guns held high. R1 is similar to any other transaction that a bank client might perform and will bring the robber closer to the money and to potential hostages, so we would like to eliminate this plan. The GRD-PL approach would remove the R1 plan allowing a robber to enter the bank with a hidden gun from the plan library. It is up to the domain designer (just like in the GRD approach) to bring about the change in the real world, such as by adding a magnetometer or a guard check in the entrance to the bank.

We begin by describing how to compute the *wcd* and *wcpd* of a given plan library. Then, we propose two alternative methods for choosing which rules can be removed from a given plan library to minimize its *wcd* and *wcpd*.

4.1 Computing Worst-Case Distinctiveness

The first challenge we face when solving the GRD-PL problem is how to compute the *wcd* of a given plan library. To this end, we first extend the notion of distinctiveness to goals. Let P_D be a library and P be a set of plans that can be derived from the plan library. We say goals $g_1, g_2 \in G$ are *distinct* given P iff $OBS(Plans(g_1) \cap P) \cap OBS(Plans(g_2) \cap P) = \emptyset$.

To illustrate, consider the plans from Figure 2. The goals of this plan library CW, MT are not distinct given all possible plans in $\{p_1, p_2, p_3, p_4\}$, since plans p_2 for achieving CW and p_4 for achieving MT both start with the sequence ⟨identification, changeSum⟩. However, the same goals given $P = \{p_1, p_2, p_3\}$ are distinct, because one can unambiguously identify the goal from the first observation.

We propose an algorithm to find the *wcd* of a plan library $P_D = \langle B, C, G, R\rangle$ by constructing plans in a top-down manner for each of the goals in P_D until we reach a point where all the goals in G are distinct. This process is shown in Algorithm 1. Initially, in the first iteration of the algorithm, P_0 is the set of plans such that each plan contains a single node labeled with a corresponding goal in G (Line 2). Let P_i denote the set of all plans in P_D that describe observation sequences of length i. The function $Distinct(P_i, G)$ receives as input a set of plans P_i and a set of goals G and returns whether the goals in G are distinct given P_i. This method is described in Algorithm 2. In this algorithm, lines 2–5 separate the different plans from P_i into classes according to their goals. These lines iterate over all plans in P_i and P_j (respectively, using the k and l indexes to enumerate the plans) and find the longest common observation sequence required to distinguish between any two plans from different goals.

At each step i of Algorithm 1, the set P_i contains all plans in P_D that describe observation sequences of length i (line 6). This procedure is performed iteratively, such that at iteration $i + 1$, we increase by one the length of the observation sequences that are described by P_i. This process terminates once goals in G are distinct given the current set of plans P_i, and the *wcd* of P_D is i.

The function $\textsc{AddOneObs}(P_i)$ expands an existing set of plans to describe an additional observation. To this end, we use the *leftmost child* definition of Geib and Goldman [16]. For a node n representing a plan p, the node c is a leftmost child of p if:

- $c \in leaves(p)$.
- There exists a rule $r \in R$ such that $r = l \rightarrow \tau \mid \prec_\tau$ where $c \in \tau$.
- In $\prec_\tau$, there is no other child that precedes l and has not already been expanded.

Given a plan p, we can expand p using a rule r iff the plan p has a leftmost child c and $r = c \rightarrow \tau \mid \prec_\tau$. Leftmost children are expanded iteratively, until exactly one observation is added to the observation sequence that describes the plan.

We can use a similar process to compute the *wcpd* of a domain. The only change will be in the function $Distinct(P_i, G)$. Instead of calculating the distinctiveness of goals, we calculate the distinctiveness of plans. We say that two plans p_i, p_j are distinct if $OBS(p_i) \cap OBS(p_j) = \emptyset$. We can extend this definition to a set of plans P_i to be distinct if for all $p_i, p_j \in P$, p_i and p_j are distinct.

The following shows that the calculation process for finding the *wcd* or *wcpd* is correct under the "bounded recursion assumption" [14, 16, 19] that limits the number of times recursive rules can be applied.

Proposition 4.1. *Algorithm 1 returns a correct wcd under the bounded recursion assumption.*

Proof. This statement is proven by showing that Algorithm 1 does not expand plans that cannot be part of the plan library, and it expands all possible plans that describe non-distinct observation sequences.

Upper bound: Let wcd^* be the correct *wcd* for a plan library P_D, and $wcd^>$ be the *wcd* returned by Algorithm 1. Assume, by contradiction, that $wcd^> > wcd^*$. This means $Distinct(P_i, G)$ returned True for the first time for some $i = wcd^> > wcd^*$. From lines 6–9 of Algorithm 2, we can infer that there is a sequence of observations O such that the following holds: First, the length of the observation sequence equals $wcd(|O| = wcd^>)$. Second, there are two plans p_1, p_2 of different goals that describe O. According to Algorithm 1 for all $i < wcd^>$, the function $Distinct(P_i, G)$ returned False. In particular, it will also return False for the set of plans of length wcd^*. But this contradicts $wcd(P_D) = wcd^*$.

Lower Bound: In this direction, we need to show that the *wcd* calculation algorithm expands all possible plans that describe non-distinct observation sequences. Suppose the algorithm returns some n that is lower than the real *wcd*, namely wcd^*. If $O = \langle o_1, \ldots, o_n\rangle$ is a non-distinct observa-

ALGORITHM 2: *Distinct*(P_n, G)

```
Input: $P_n$: the current set of plans.
Input: $G$: the set of goals in the plan library.
Output: isDistinct: whether the goals are distinct.
1  $isDistinct \leftarrow True$
2  foreach $g \in G$ do
3      $P_g \leftarrow Plans(g) \cap P_n$
4      if $P_g = \emptyset$ then
5          return $False$
6  for $i, j \in G, i \neq j$ do
7      for $p_i^k \in P_i, p_j^l \in P_j$ do
8          if $\exists O \in \{OBS(p_i^k) \cap OBS(p_j^l)\} \wedge (|O| = n)$ then
9              $isDistinct \leftarrow False$
10 return $isDistinct$
```

tion sequence with $n \geq 1$, then this means that there are p_1, p_2 such that $goal(p_1) \neq goal(p_2)$ and $O \in OBS(p_1) \cap OBS(p_2)$. Hence, from the first observation o_1 that can be described by p_1 and p_2, the *Distinct* function will return False (Algorithm 2, line 5). This will be the returned value for the *Distinct* function until the $n + 1$th observation. Hence, the algorithm will not return a *wcd* value before the $n + 1$ iteration, and the *wcd* will be at least n. □

The same proof can be easily changed to show that the same proposition also holds for the *wcpd* calculation.

4.2 Optimizing Plan Library Design

Using the algorithm above for computing the *wcd* and *wcpd*, we now present two algorithms for solving the GRD-PL problem, and then we describe how they can be easily adapted to solve the PRD problem. For ease of notation, we denote by $p \in P_D$ that plan library P_D has a goal g for which $p \in Plans(g)$.

The first algorithm we propose, called ReduceBF, uses a brute-force method to search for the minimal set of rules to remove to minimize the *wcd* in GRD-PL. Algorithm 3 lists the pseudo code of ReduceBF. It gradually removes rules from the given plan library, starting from examining all removals of a single rule, then removals of two rules, and so on. It halts only when either (1) there are no more rules that can be removed, meaning all combinations of rules that can be removed while still enabling achieving all goals were tested, or (2) we have reached a plan library with a *wcd* of 0.

For example, consider the plan library presented in Figure 2. ReduceBF will attempt to remove all single rules from the plan library. Suppose that each of the plans in the plan library can be created by use of a single rule. After the removal of the rule that created plan p_4, the *wcd* of the plan library is 0, and the algorithm will stop. In the empirical section, we show that this algorithm can find reductions of the given plan library that have a significantly low *wcd*, for plan libraries from the PR literature.

In the worst case, ReduceBF will consider all possible combinations of rules from the original plan library (line 3). In one of our settings in the empirical section, the Monroe settings, there are 89 rules in the plan library. This means that a complete scan of all possible reductions requires 2^{89}

ALGORITHM 3: *ReduceBF*(PL, G)

```
Input: $PL$: the current plan library.
Input: $G$: the set of goals in $PL$.
Output: $PL_{new}$: The new plan library with lower $wcd$ value.
1  $wcd_{best} \leftarrow wcd(PL)$
2  $PL_{best} \leftarrow PL$
3  foreach $R_{curr} \in 2^R$ do
4      if $\forall g \in G$ there is a plan $p \in PL$ that does not use $R_{curr}$ then
5          $PL_{curr} \leftarrow$ Reduce$(PL, R_{curr})$
6          $wcd_{curr} \leftarrow wcd(PL_{curr})$
7          if $wcd_{curr} < wcd_{best}$ then
8              $wcd_{best} \leftarrow wcd_{curr}$
9              $PL_{best} \leftarrow PL_{curr}$
10         if $wcd_{best} == 0$ then
11             return $PL_{best}$
12 return $PL_{best}$
```

combinations, which is infeasible. To address this efficiently, we now introduce the ReduceCBS algorithm that builds on the following definition and observation:

Definition 10 (Conflicting Plans). If there is an observation sequence O that can be described by plans p_1 and p_2, then we say that p_1 and p_2 are in conflict. If p_1 and p_2 are defined by a plan library P_D and $wcpd(p_1, p_2) = wcd(P_D)$, then we say that p_1 and p_2 have a maximal conflict in P_D.

Lemma 2. *Let p_1 and p_2 be a pair of plans that have a maximal conflict in P_D. To obtain a reduction of P_D, at least one rule that is used in p_1 or p_2 (or both) needs to be removed.*

Proof. Let p_1 and p_2 be a pair of plans that have a maximal conflict in P_D, and let P'_D be a reduction of P_D. By contradiction, assume that P'_D contains all the rules used by plans p_1 and p_2. This means that plans p_1 and p_2 can still be generated in P'_D. Since these plans have a maximal conflict in P_D, it means that there exists an observation sequence O that can be described by both p_1 and p_2 such that $|O| = wcd(P_D)$. Consequently, $wcd(P'_D) = |O| = wcd(P_D)$, negating the assumption that P'_D is a wcd-reduction. □

ReduceCBS exploits Lemma 2 by considering to remove only rules that are used by conflicting plans. ReduceCBS is a best-first search algorithm that works as follows: Initially, the openlist contains only the original plan library. In every iteration henceforth, a plan library (denoted PL_{curr}) is removed from the openlist. If this plan library contains a pair of plans that is in conflict (Definition 10), then we iterate over every rule used by these plans, and create for every such rule an identical plan library that is missing the rule. This reduced plan library is inserted to the openlist, to be considered later for further reduction. This search process continues until either we get to a plan library that has a wcd of zero, or until the openlist is empty, which means there are no more rule combinations to remove. In our implementation, the openlist was sorted according to the number of removed rules, i.e., extracting from the openlist in every iteration the plan library obtained by removing the minimal number of rules from P_D. Other techniques for sorting the openlist are also possible.

In the example of Figure 2, we know that the wcd of the plan library used to create the plans is 2. Plans p_2 and p_4 have a conflict with $wcd = 2$. After calling ReduceCBS, we will remove plan p_4 from the plan library. At this point the wcd of the plan library is 0, because a single observation is

ALGORITHM 4: *ReduceCBS*(PL, G)

```
Input: $PL$: the current plan library.
Input: $G$: the set of goals in $PL$.
Output: $PL_{new}$: The new plan library with lower $wcd$ value.
1  $wcd_{best} \leftarrow wcd(PL)$
2  $PL_{best} \leftarrow PL$
3  $open \leftarrow PL$
4  while $open \neq \emptyset \wedge wcd_{best} \neq 0$ do
5      $PL_{curr} \leftarrow open.pop()$
6      $\langle p_1, p_2 \rangle \in$ getConflict$(PL_{curr})$
7      $R_{curr} \leftarrow$ rules$(p_1) \cup$ rules$(p_2)$
8      foreach $r_{sub}$ in $R_{curr}$ do
9          if $\forall g \in G$ there is a plan $p \in PL_{curr}$ that does not use $r_{sub}$ then
10             $PL_{sub} \leftarrow$ reduce$(PL_{curr}, \{r_{sub}\})$
11             $wcd_{sub} \leftarrow wcd(PL_{sub})$
12             if $wcd_{sub} < wcd_{best}$ then
13                 $wcd_{best} \leftarrow wcd_{sub}$
14                 $PL_{best} \leftarrow PL_{sub}$
15             $open.push(PL_{sub})$
16 return $PL_{best}$
```

sufficient to identify the goal. Note that removing p_2 will not be enough, since plan p_1 still conflicts with plan p_4 with $wcd = 1$.

Observe that ReduceCBS is advantageous compared to ReduceBF even if it is not possible to reduce the *wcd* to zero. This is because ReduceCBS does not consider the entire space of possible rule reductions, as it only considers rules that exist in conflicting pair of plans. Indeed, in the empirical section below, we show that ReduceCBS significantly reduces the number of rules needed to consider to optimize the *wcd* value. For example, in one of our settings (Monroe) the original plan library had a *wcd* of three. ReduceCBS first tries to reduce the *wcd* of these two conflicting plans. This means it will need to consider only combinations of rules that created these two plans (specifically in the Monroe example, it involved 16 rules instead of 89).

Next, we establish some theoretical properties of ReduceBF and ReduceCBS. Both algorithms are designed to solve the GRD-PL problem (Definition 7). Thus, a GRD-PL algorithm is called *sound* if it returns a plan library that is a *wcd*-reduction of the original plan library. A GRD-PL algorithm is called *complete* if it is guaranteed to find a *wcd*-reduction if such exists. Finally, a GRD-PL algorithm is called *optimal* if it returns an optimal *wcd*-reduction, i.e., a reduction with the smallest *wcd* possible where all goals are reachable. ReduceBF considers every possible reduction of the original plan library P_D. It returns the plan library it had found that had the minimal *wcd*. Thus, ReduceBF is clearly sound, complete, and optimal.

To prove the same for ReduceCBS, we first establish the following observation.

Observation 1. *Let $P_D = \langle B, C, G, R \rangle$ be the plan library outputted by ReduceCBS, let $P'_D = \langle B, C, G, R' \rangle$ be a possible wcd-reduction of P_D, and let $r \in P_D$ be a rule that is not used by any plan in a maximal conflict in P_D. If $r \notin P'_D$, then the plan library $P''_D = \langle B, C, G, R' \cup \{r\} \rangle$ is a reduction of P_D.*

In other words, if P'_D is a reduction of P_D and some rule r that is not used by plans in conflict that were removed from P_D, then P'_D with that rule is also a reduction of P_D.

Proof. If P''_D is not a *wcd*-reduction of P_D, then $wcd(P''_D) = wcd(P_D)$. This means there are two plans p_1 and p_2 defined by P''_D, an observation sequence O such that $wcd(P_D) = |O|$, and O is described by p_1 and by p_2. Since r is not used by neither p_1 nor p_2, then p_1 and p_2 can also be generated with P'_D and thus $wcd(P'_D) = wcd(P_D)$, negating the assumption that P'_D is a *wcd*-reduction of P_D. □

Theorem 4.2. *ReduceCBS (Algorithm 4) is sound, complete, and returns an optimal solution to GRD-PL.*

Proof. For every *wcd*-reduction P'_D, we can apply Observation 1 iteratively to add to it all the rules that are not used by a plan in P_D that has a maximal conflict. This will result in a *wcd*-reduction that contains only rules that are used by plans in P_D that have a maximal conflict. This *wcd*-reduction, in turn, is found by ReduceCBS as it considers *wcd*-reductions for rules used in plans with maximal conflict. Thus, ReduceCBS is complete. Soundness is easily established as ReduceCBS never removes a rule that prevents achieving a goal.

Next, we prove the optimality of the solution returned by ReduceCBS. The openlist maintained by the ReduceCBS algorithm contains a set of plan libraries, derived from the original plan library by removing some of its rules. For a given plan library P, let $\Pi(P)$ be the set of all reductions of P. To prove that ReduceCBS is optimal, we now prove that in every iteration of ReduceCBS, either an optimal plan library P^*_D has already been found or there exists a plan library P'_D in the openlist such that $P^*_D \in \Pi(P'_D)$. We prove this by induction on the number of iterations of ReduceCBS.

In the first iteration, the openlist contains only the original plan library P_D. Since P^*_D is a reduction of P_D, then by definition, $P^*_D \in \Pi(P_D)$.

Now assume that the induction assumption holds for the first n iterations of ReduceCBS. This means that after the first n iterations, either we already found an optimal plan library or the openlist contains a plan library P'_D such that $P^*_D \in \Pi(P'_D)$. If it is the former case, then we are guaranteed that ReduceCBS will return an optimal solution.

Consider the latter case, i.e., where in the nth iteration of ReduceCBS there is a plan library P'_D in the openlist such that $P^*_D \in \Pi(P'_D)$. In every iteration of ReduceCBS, a single plan library is popped out of openlist (line 5 in Algorithm 4). If P'_D was popped out of the openlist in the nth iteration, then P'_D will still be in the openlist in the next iteration the induction assumption holds. Now, consider the case where P'_D is popped from open in the nth iteration. P^*_D is a reduction of P'_D, and thus there exists at least one rule r in P'_D that is used by a plan with a maximal conflict and that rule is not in P^*_D (Observation 1). Let P''_D be a plan library that results from removing r from P'_D. By definition, $P^*_D \in \Pi(P''_D)$, P''_D was generated from P'_D by only removing r and r is not in P^*_D. In addition, ReduceCBS will generate P''_D after popping P'_D out of the openlist and add it to the openlist, since it is generated from P'_D by removing a rule used by a plan that has a maximal conflict. Thus, induction assumption holds also for the $n + 1$ iteration of ReduceCBS, as required. □

Historically, ReduceCBS was inspired by the Conflict-based Search (CBS) algorithm [39]. CBS is an algorithm for solving the multi-agent pathfinding (MAPF) problem. In MAPF, there is a set of agents, each having a start and goal location and the task is to find a path for each agent so that the agents do not collide. CBS solves this problem by constructing a plan for each agent individually, then identifying conflicts between the paths, and resolving them by replanning for the conflicting agents. ReduceCBS is similar to CBS in that both algorithms search in the space of conflicts and how to resolve them: In CBS, a conflict in a pair of plans is a potential collision between the agents following these plans, and in ReduceCBS, a conflict in a pair of plans corresponds to an observation sequence that can be described by both plans and its length is equal to the plan library's *wcd*.

4.3 From GRD-PL to PRD

Adapting the brute-force algorithm (Algorithm 3) and ReduceCBS (Algorithm 4) to solve PRD is straightforward. The only changes are to compute the *wcpd* instead of the *wcd* of the generated plan libraries, and to halt the search when the *wcpd* cannot be reduced any more instead of when the *wcd* cannot be reduced anymore. Theorem 4.1 can be easily adapted to prove that the PRD versions of the brute-force algorithm ReduceCBS are sound, complete, and return an optimal solution to PRD.

We note that both algorithms when adapted to PRD may continue to search for rules to reduce even when the *wcd* is 0, since we might still be able to reduce the *wcpd*. In the example of the plan library presented in Figure 2, after the removal of p_4, the *wcd* of the plan library is 0, but its *wcpd* will be 1. The algorithms can then continue to remove the rule that led to plan p_2 (or p_1) from the plan library. At this point, the *wcpd* of the plan library will be 0, and the algorithm will stop.

5 COMPLEXITY ANALYSIS

This section is constructed as follows: First, there is a complexity analysis for the *wcd* and *wcpd* calculation presented in Algorithms 1 and 2, and then there is another analysis for the different reduction algorithms presented in the article—the Brute-force approach in Algorithm 3 and the Conflict-based Search in Algorithm 4.

5.1 Metrics Calculation

The cost of computing the *wcpd* of a domain is higher than the cost of computing its *wcd*. There are two parts for this computation: first, the cost of expanding a level of plans that describe observation sequences of length n; and second, the cost of deciding whether another level should be expanded.

The first level requires us to produce $|G|$ nodes, one node per possible goal. Expanding the search tree from the $i-1$th level to the ith level, requires us to create a path of child nodes for all leftmost children to a new observation. This is done for every plan from level $i-1$.

For each node n describing a plan p, and for every leftmost child c, we define the set of rules $\{r' \in R \text{ s.t. } r' : c \rightarrow \tau \mid O\}$, where c is the left-hand side of the rule (that can be decomposed into the sequence τ under the ordering constraints in O). Given c, we need to create $|r'|$ new nodes and each node is a plan describing the same observations as p plus one new observation. The construction of each such node requires us to connect the new observation to the plan. This connection requires us to apply at most d rules, where d is the maximum length of a path from the leftmost child c to the newly added observation. Let τ_{max} be the rule with the longest right-hand side, then if we define the number of nodes in the $i-1$th level to be n_{i-1}, then at the worst case, we will have $n_{i-1} \cdot d \cdot (|\tau_{max}|-1)^{i-1} \cdot |r|$ nodes in the ith level, as we will have all possible leaves as leftmost children. In total, expanding k levels of the search tree will require us $O(|G| \cdot d \cdot \Sigma_{i=1}^{k} (|\tau_{max}|-1)^i \cdot |r|^{i-1})$.

Next, when we reach a level with k plans, we need to see if these nodes represent plans that describe a mutually exclusive sets of observations. Let $G_1, \ldots, G_n$ be the set of goals in the plan library, and let $P_1, \ldots, P_n$ be the number of different plans for each goal, i.e., P_i is the number of plans for goal G_i. Let X be the cost of checking the longest observation sequence described by a pair of plans. According to Definition 4, $X \leq \tau_{max}$. The runtime of computing the longest observation sequence between different goals for a complete plan library, denoted T_{wcd}, is $T_{wcd} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} P_i \cdot P_j \cdot X$.

Similarly, the runtime of computing the longest observation sequence between all possible plans in a complete plan library, denoted $T_{wcpd} = \sum_{p_i, p_j \in L_D} p_i \cdot p_j \cdot X$, where L_D is the set of all possible plans in the plan library. In the extreme case where each goal has exactly one plan that can be executed, T_{wcd} will be equal to T_{wcpd}. In all other cases, T_{wcpd} will be larger.

Finally, we need to combine the results of the two parts to show the complete calculation time of expanding the search tree, with the distinctiveness check at each level. This combination is the complexity of calculating the *wcd* of a plan library: $O(|G| \cdot d \cdot \Sigma_{i=1}^{k}(T_{wcd} \cdot |\tau_{max}|)^{i} \cdot |r|^{i-1})$. Similarly, the complexity of calculating the *wcpd* of a plan library is: $O(|G| \cdot d \cdot \Sigma_{i=1}^{k}(T_{wpcd} \cdot |\tau_{max}|)^{i} \cdot |r|^{i-1})$. Note that the k value, representing the number of levels required to expand, can be larger when calculating the *wcpd* than when calculating the *wcd*, as longer observation sequences might need to be considered (According to Theorem 1).

According to Theorem 1, the *wcd* of a plan library is smaller than or equal to the *wcpd*. This result highlights that computing the *wcd* can be done faster than computing the *wcpd*. This advantage is mainly expressed when *wcpd* and *wcd* are very different.

5.2 Reduction Calculation

The main loop in the brute-force algorithm (Algorithm 3) is repeated $2^{|P|}$ times, since it considers all possible rule combinations. In each iteration of the main loop, we calculate the new *wcd* or *wcpd* value. So, the overall complexity of the brute-force algorithm for the *wcd* calculation is $O(2^{|P|} \cdot |G| \cdot d \cdot \Sigma_{i=1}^{k}(T_{wcd} \cdot |\tau_{max}|)^{i} \cdot |r|^{i-1})$.

The ReduceCBS algorithm only considers removing rules that construct plans that are in conflict with the current *wcd*. In the worst case, this can be all rules, and the complexity of the ReduceCBS algorithm will be the same as the brute-force algorithm. However, if there are $1, \ldots, c$ conflicts that need to be resolved, and each conflict i is constructed from r_i rules, then the number of combinations to consider is $O(2^{\Sigma_{i=1}^{c} r_i}) \leq O(2^{|P|})$.

Regarding both reduction algorithms, the naive implementation of the reduction process requires us to calculate the *wcd* for every candidate plan library. This, in turn, requires us to generate for scratch all possible plans of length $\leq$ *wcd* for each such plan library. If memorization is used, then the initial *wcd* calculation creates a data structure with all plans that will be needed in future calculations. Using this data structure, the time required to calculate all combinations of rules in the plan library is $O(|R| \cdot d \cdot k \cdot (|\tau_{max}| - 1)^{i-1} \cdot |r|)$, which is basically the cost of traversing the data structure times the number of subsets of rules we need to consider.

6 EMPIRICAL EVALUATION

We evaluate our work in three common settings from the plan recognition literature. The first is the Monroe, a disaster management setting [7]. The plan library for this settings includes 30 basic actions, 70 complex actions, 10 possible goals, and 89 rules. Actions in this domain are partially ordered. The second setting is the Soccer settings used by Avrahami-Zilberbrand and Kaminka [4]. This domain has 7 basic actions, 10 complex actions, 3 possible goals, and 13 rules. Actions in this setting are fully ordered. The third is a simulated setting used to evaluate plan recognition algorithms in several prior works [15, 19, 30]. We used two types of simulated settings, named Simulated1 and Simulated2. Both settings include 30 complex actions, 2 possible goals and 50 rules with partial ordering. Simulated1 has 100 basic actions, which is similar to the configuration used in the prior work mentioned above. Simulated2 has 20 basic actions (a smaller vocabulary), which makes the settings more ambiguous [25], as there can be more possible plans for completing each goal.

All of the grammars used in these settings exhibit the property that a rule in the grammar cannot have more than one basic action in its left-hand side. This rule is required by Algorithm 2 to increase the level of each plan by one observation. Any grammar can be converted to exhibit this property [16].

Table 2. *wcd* and *wcpd* Values, Runtime, and Comparisons Performed for Empirical Settings

Settings/	Value		Time		Comparisons	
Measure	*wcd*	*wcpd*	*wcd*	*wcpd*	*wcd*	*wcpd*
Soccer	2.00	3.00	0.01	0.03	11	78
Monroe	3.00	17.00	0.31	3.83	179	25,425
Simulated1	0.82	3.26	0.16	1.09	1,501	1,319,412
Simulated2	2.69	3.95	0.90	1.06	672	1,378,114

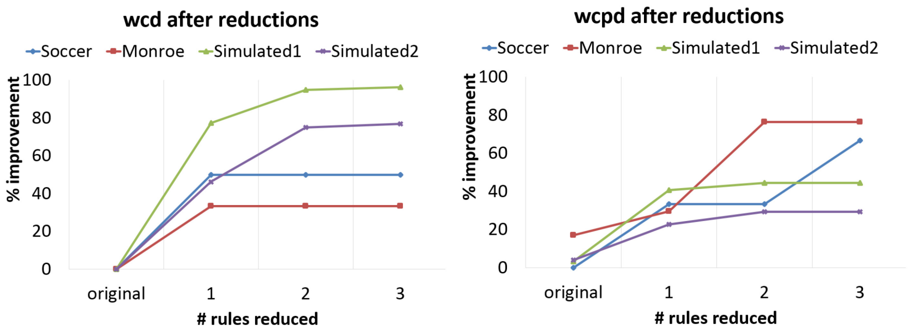

Fig. 3. Improvement in *wcd* (left) and *wcpd* value (right) for Simulated2 settings using brute-force reductions method.

Table 2 shows the original *wcd* and *wcpd* values of each settings, their computation time (in seconds) and the number of comparisons performed to calculate them. All experiments were run on a commodity I7 computer. As shown in the table, for all domains the value of the *wcpd* and is greater than the value of the *wcd*, which follows Theorem 1. The *wcpd* of the Monroe settings is significantly larger than its *wcd*. Specifically, it takes 17 observations to distinguish between two possible plans for providing temporary heat for survivors. In addition, computing the *wcpd* requires more time due to the number of plans we need to compare for distinctiveness. As the number of observations grows, so does the number of plans that can describe them.

Interestingly, the number of plan comparisons for computing the *wcpd* is orders of magnitude larger than the number of comparisons for computing the *wcd*. This is due to the following factors. First, by Theorem 1 *wcpd* is at least as high as the *wcd*. Second, computing the *wcpd* requires us to compare many plans with the same goal, while for the *wcd*, we only need to compare plans across different goals.

6.1 Brute-Force Reductions

Figure 3 shows how the *wcd* and *wcpd* values can be improved by allowing reductions of at most n-rules. The x axis is the maximum number of rules that can be reduced. The y axis shows the improvement (in percentages) in the *wcd* or *wcpd* values. A 100% improvement can only be reached if the *wcd* or *wcpd* can be reduced to 0. This is the reason the *wcd* of the Soccer and Monroe do not improve beyond a certain point, which represents the upper bound of improvement for these settings. Moreover, the runs presented in this graph had a 30min timeout. This means that even when there is an option to reduce the *wcd* and *wcpd* values of the domain further, the reduced set of rules cannot be reached within the 30min timeout. This can be seen in the plateau for the *wcpd*'s runs for the Simulated2 settings.

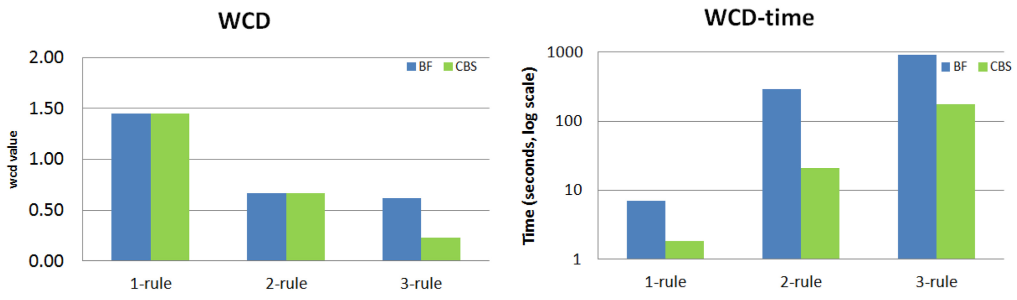

Fig. 4. The *wcd* values and computation times using Brute-force and CBS Reductions.

Together, these results highlight the benefit of our approach, in that less observations are required to unambiguously recognize agent's goals. In most settings, even a 1-rule reduction is effective in reducing both *wcd* and *wcpd*. Removing more rules reduces the *wcd* and *wcpd*, in most cases to the best possible outcome. Last, these graphs emphasize that the *wcd* and *wcpd* reductions behave differently—*wcd* reductions have different effects than *wcpd* reductions. For example, for the Monroe settings, the improvement in *wcpd* was higher than that of the *wcd*. For the Soccer settings, the *wcd* reduction reaches a plateau after removing a single rule, while the *wcpd* process continues to improve.

6.2 Conflict-Based Search Reductions

Next, we measure the impact of using the ReduceCBS algorithm. The results presented in Figure 4 are for Simulated2 only, but other settings show similar trends. Each couple of bars show the values for BF compared to ReduceCBS (CBS). The x-axis is the number of rule combinations that were reduced (1, 2, or 3). The y axis shows the absolute *wcd* value (left), *wcd*-time is measured in seconds and presented in log scale.

Theoretically, given enough time, BF and CBS will output the same solutions, however, CBS is significantly faster than BF, and has practical implications: Given the 30min timeout, the CBS algorithm can outperform brute force. In the three-rule reductions, CBS manages to find solutions that BF could not due to the given time limit.

We measured how many instances were completed in 30min on CBS: In the Soccer setting, in all instances best possible *wcd* and *wcpd* reductions were achieved before the 30min deadline. In the Monroe setting, the best possible *wcd* reductions were achieved for all rule combination sizes, but (possibly) not the optimal *wcpd*, as it was timed-out. For Simulated1, CBS was able to complete all runs for the *wcd* reductions and 17 out of 100 runs for the *wcpd* reductions. For Simulated2, CBS was able to complete all runs for the *wcd* reductions and 5 out of 100 runs for the *wcpd* reductions.

To illustrate the benefit of the ReduceCBS algorithm, the lowest number of rules in a plan library that was returned by the ReduceCBS algorithm within 30min was 28, meaning a reduction of 22 rules from the plan library. If we used the brute-force method instead, then this means we had to reach computations of 2^{22} combinations until reaching this reduction.[2]

7 RELATED WORK

A body of work has focused on domain design as means to address the ambiguity problem. Keren et al. [20] introduced the notion of worst-case distinctiveness (*wcd*) as a measure of the difficulty of performing goal recognition. They assumed a STRIPS domain representation. The *wcd* of a

[2]Note that for those instances that have not been completed within the time limit, we cannot tell whether the obtained *wcpd* value is optimal.

problem is the maximum length of a prefix of an optimal sequence of actions an agent may take within a system before it becomes clear at which goal it is aiming. The objective in GRD is then to minimize the *wcd* without constraining the actor from achieving all goals. Later works improved these approaches to account for non-optimal agents [21] and non-observable actions [22]. They generalized their work to reason about various types of utility-maximizations by formulating all possible environments in general, such that the *wcd* becomes a single possible metric, and others can be defined as well on a set of environments [24].

Wayllace et al. [45] defined GRD with stochastic agent action outcomes and Son et al. [41] presented an approach to solving GRD using Answer Set Programming (ASP). Another recent work extends GRD to settings where the observer has partial observability of the agent's actions, while the agent is assumed to have full observability of its environment [23]. Wayllace et al. [44] introduced a new metric, expected-case distinctiveness (*ecd*), that puts weights on the different goals. All these works assume a domain representation based on STRIPS as presented in References [32, 40] and were not applied to hierarchical plans. We showed in Section 3 that reasoning about plan hierarchies within the GRD paradigm is not a straightforward extension to their theory and requires new types of methods.

Ramirez and Geffner [34] provided an approach to compile plan libraries (represented as AND-OR trees) into STRIPS representation. In theory, one can solve the GRD-PL problem by solving the corresponding GRD problem in STRIPS space. There are several issues with this approach. First, the resulting STRIPS representation can be prohibitively large. To illustrate, one of the plan libraries used by Ramirez and Geffner with 85 nodes was converted to a STRIPS representation with 800 actions, while a plan library with 251 nodes was converted to a STRIPS problem with over 10,000 actions. In general, the complexity of the GRD process is exponential in the size of the corresponding STRIPS representation. Second, there is no guarantee that the STRIPS representation can be compiled back to a valid plan library, and the resulting STRIPS representation may not be easily interpretable by people. Thus it is important to study the plan recognition design problem directly in the context of plan libraries. Moreover, it is sometimes the case that plan libraries are chosen due to their descriptive representation (e.g., to visualize the hierarchical nature of the agent's activities). In such cases, even if the compilation to STRIPS is feasible, there is still a preference to use plan libraries.

Plan-library-based plan recognition derives from the analogy between plan libraries and grammars. In particular, previous works about parsers tried to design grammars and parsers that will be able to disambiguate between different parsings after a constant number of observed tokens. Rozenkrantz and Stearns [35] and Aho et al. [1] showed how to construct a linear parser without backtracking (e.g. until all ambiguity of recognizing the sentence structure can be solved). These works do not reason about plan libraries, which can involve an interleaving of actions.

Geib and Goldman [15, 16] studied the relationship between recognition ambiguity in a domain and the ordering constraints over rules in the domain. They showed that the number of plans can be reduced when ordering constraints enforce the first action of a rule.

8 CONCLUSION AND FUTURE WORK

This article presented two new approaches for plan- and goal-recognition domain design: Plan Recognition Design (PRD) is the task of designing a domain using plan libraries to facilitate fast identification of an agent's plan. Goal Recognition Design with Plan Libraries (GRD-PL) extends Goal Recognition Design [20] approach to using plan libraries as a domain representation.

For each of these approaches the article defines a new measure—*wcd* and *wcpd*, respectively—and provides an anytime algorithm for computing them and optimizing the recognition domain accordingly. We show that the *wcd* of GRD-PL is a lower bound to the *wcpd* for PRD, and they are equal under certain conditions. Therefore, solving the (simpler) GRD-PL problem may at times yield an optimal solution for the PRD problem. We evaluated our approach in three known hierarchical planning domains from the literature using the new measures.

Our empirical work provides a novel evaluation for known plan libraries as well as how plan libraries can be modified in the different settings to reduce their *wcd* and *wcpd*. The empirical results emphasize the following points: First, in all the settings the *wcd* and *wcpd* were not minimal. Second, the reduction process for these settings enabled faster disambiguation of the hypotheses. Even a reduction of a single rule reduced the *wcd* and *wcpd* values. Moreover, the ReduceCBS algorithm allowed us to reach better results in a given time frame.

Last, we investigated the relation between the *wcd* and *wcpd* and the orientation of the reduction algorithms to either metric.

The results in this article have insight for domain designers using plan recognition to infer agents' activities. They provide a way of measuring the ambiguity that is inherent in a domain using worst-case distinctiveness measures and also algorithms for reducing this ambiguity. As we show empirically, the approach was able to reduce the worst-case distinctiveness of real-world domains and improve the efficiency of plan recognition on these domains.

We are currently extending our work in several directions. First, removing rules from plan libraries might put heavy restrictions on an agent's actions. We are considering other, less restrictive, manipulations on the domain, such as adding ordering constraints on actions.

Within the reduction algorithms of Section 4 is a tradeoff in the decision of which rule combinations to remove. Consider the scenario containing 3 plans, where p_1, p_2 are identical except for an action at the leaves of the plan tree, while p_3 is different from p_1 and p_2 is close to the root. Removing the rule that creates the difference between p_1 and p_2 is more specific, because it can restrict the agent from achieving one of these plans (but not the other). In contrast, removing the rule that creates the difference between p_3 and p_1, p_2 can restrict the agent from completing both plans p_1 and p_2. The extent to which rules restrict the agent's activities depend on the "specificity" of the rule. We intend to formally study this tradeoff in future work. We are also designing heuristics for ordering which rule combinations to consider for removal from the plan libraries.

Another interesting tradeoff is the one between computation time and the quality of the GRD-PL and GRD approach (as measured by*wcd* and *wcpd*). As seen in Section 6.1 this effect is stronger for *wcd* reductions than for *wcpd* reductions. To reason about this tradeoff, we will modify the GRD-PL and PRD approaches to anytime algorithms.

APPENDIX

A IMPROVEMENTS

There are many ways and approaches for improving the efficiency of the algorithms described in this article. Here, we present several approaches to improve the calculation of the *wcd* and *wcpd* of a plan library. We divide these improvements into two types: (1) Algorithmic improvements and (2) Technical improvements. The first type is meant to improve the general process of the calculation, and the latter provides specific implementation suggestions.

A.1 Algorithmic Improvements

Algorithm 1 provides a general template for computing the *wcd*. Keren et al. [20] showed that if an observations sequence O is distinct, then any observation sequence O' such that O is a prefix of O' is distinct as well. This result can be used to enforce two improvements to this process:

Discarding distinctive goals. If the set of plans $P_i^1 = P_i \cap plans(g_1)$ is distinct from the plans of all other goals, then we can stop expanding the plans in P_i^1.

Discarding distinctive plans. For some iteration i and a set of plans P_i, if a plan $p \in P_i$ is distinct from all other plans, then we can stop expanding p. This means that all $OBS(p)$ are distinct, and according to the Corollary, there is no sequence that can be added to $OBS(p)$ that will make it non-distinct, so we can stop expanding this plan.

A.2 Technical Improvements

In our implementation, we devised a couple of optimizations to make the distinct process more efficient. When looking at a plan $p \in P_i$, we can skip checking if it is distinct from other plans if:

(1) $OBS(p) < wcd$. If p describes sequences shorter than the current wcd, then it cannot be used to show a larger wcd.
(2) $OBS(p) \leq wcd$ and $\neq isDistinct$. If p describes sequences that are no longer than the current wcd and there is already some other evidence to a sequence that is at least as long as the longest sequence in $OBS(p)$, then it cannot be used to show a larger wcd.

While these improvements do not change the worst-case runtime of the algorithm, it did provide a significant improvement in practice.

ACKNOWLEDGMENTS

R. M. is a recipient of the Ministry of Science fellowship for Research in Applied Science and Engineering.

REFERENCES

[1] A. V. Aho, S. C. Johnson, and J. D. Ullman. 1975. Deterministic parsing of ambiguous grammars. *Commun. ACM* 18, 8 (1975), 441–452.
[2] O. Amir. 2014. Information sharing for care coordination. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence.*
[3] O. Amir and Y. Gal. 2013. Plan recognition and visualization in exploratory learning environments. *ACM Trans. Interac. Intell. Syst.* 3, 3 (2013), 16:1–23. DOI: https://doi.org/10.1145/2533670.2533674
[4] D. Avrahami-Zilberbrand and G. A. Kaminka. 2005. Fast and complete symbolic plan recognition. In *Proceedings of the International Joint Conference of Artificial Intelligence (IJCAI'05)*, Vol. 14.
[5] R. Barták, A. Maillard, and R. C. Cardoso. 2018. Validation of hierarchical plans via parsing of attribute grammars. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling.*
[6] G. Behnke, D. Höller, and S. Biundo. 2017. This is a solution! (...but is it though?) Verifying solutions of hierarchical planning problems. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17).*
[7] N. Blaylock and J. Allen. 2005. Generating artificial corpora for plan recognition. In *Proceedings of the International Conference on User Modeling*. Springer, 179–188.
[8] N. Blaylock and J. Allen. 2006. Fast hierarchical goal schema recognition. In *Proceedings of the National Conference on Artificial Intelligence*, Vol. 21, 796.
[9] B. G. Buchanan, E. H. Shortliffe et al. 1984. *Rule-Based Expert Systems.* Vol. 3. Addison-Wesley Reading, MA.
[10] M. S. Fagundes, F. Meneguzzi, R. H. Bordini, and R. Vieira. 2014. Dealing with ambiguity in plan recognition under time constraints. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS'14).* International Foundation for Autonomous Agents and Multiagent Systems, 389–396.
[11] R. E. Fikes and N. J. Nilsson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artific. Intell.* 2, 3–4 (1971), 189–208.
[12] R. G. Freedman and S. Zilberstein. 2017. Integration of planning with recognition for responsive interaction using classical planners. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI'17).* 4581–4588.
[13] Y. Gal, E. Yamangil, A. Rubin, S. M. Shieber, and B. J. Grosz. 2008. Towards collaborative intelligent tutors: Automated recognition of users' strategies. In *Proceedings of the Conference on Intelligent Tutoring Systems (ITS'08).*

[14] C. W. Geib and R. P. Goldman. 2011. Recognizing plans with loops represented in a lexicalized grammar. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI'11).*

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

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

[17] Christopher W. Geib and Pavan Kantharaju. 2018. Learning combinatory categorial grammars for plan recognition. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI'18).*

[18] F. Kabanza, P. Bellefeuille, F. Bisson, A. R. Benaskeur, and H. Irandoust. 2010. Opponent behaviour recognition for real-time strategy games. *Plan Activ. Intent Recogn.* 10, 05 (2010).

[19] F. Kabanza, J. Filion, A. R. Benaskeur, and H. Irandoust. 2013. Controlling the hypothesis space in probabilistic plan recognition. In *Proceedings of the International Joint Conference of Artificial Intelligence (IJCAI'13).* 2306–2312.

[20] S. Keren, A. Gal, and E. Karpas. 2014. Goal recognition design. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS'14).*

[21] S. Keren, A. Gal, and E. Karpas. 2015. Goal recognition design for non-optimal agents. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI'15).* 3298–3304.

[22] S. Keren, A. Gal, and E. Karpas. 2016. Goal recognition design with non-observable actions. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI'16).*

[23] S. Keren, A. Gal, and E. Karpas. 2016. Privacy preserving plans in partially observable environments. In *Proceedings of the International Joint Conference of Artificial Intelligence (IJCAI'16).*

[24] S. Keren, L. Pindea, A. Gal, E. Karpas, and S. Zilberstein. 2017. Equi-reward utility maximizing design in stochastic environments. In *Proceedings of the International Joint Conference of Artificial Intelligence (IJCAI'17).*

[25] M. C. MacDonald, N. J. Pearlmutter, and M. S. Seidenberg. 1994. The lexical nature of syntactic ambiguity resolution. *Psychol. Rev.* 101, 4 (1994), 676.

[26] J. Maraist. 2016. String shuffling over a gap between parsing and plan recognition. In *Proceedings of Plan, Activity and Intent Recognition (PAIR) Workshop on the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18).*

[27] R. Mirsky and Y. Gal. 2016. SLIM: Semi-Lazy inference mechanism for plan recognition. In *Proceedings of the International Joint Conference of Artificial Intelligence (IJCAI'16).*

[28] R. Mirsky, Y. Gal, and S. M. Shieber. 2017. CRADLE: An online plan recognition algorithm for exploratory domains. *ACM Trans. Intell. Syst. Technol.* 8, 3 (2017), 45.

[29] R. Mirsky, Y. Gal, and D. Tolpin. 2017. Session analysis using plan recognition. In *Proceedings of User Interfaces and Scheduling and Planning (UISP'17) at the International Conference on Automated Planning and Scheduling (ICAPS'17).*

[30] R. Mirsky, R. Stern, Y. Gal, and M. Kalech. 2016. Sequential plan recognition. In *Proceedings of the International Joint Conference of Artificial Intelligence (IJCAI'16).*

[31] X. Qin and W. Lee. 2004. Attack plan recognition and prediction using causal networks. In *Proceedings of the 20th Annual Computer Security Applications Conference.* IEEE, 370–379.

[32] M. Ramırez and H. Geffner. 2009. Plan recognition as planning. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI'09).*

[33] Miquel Ramırez and Hector Geffner. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI'10).* Citeseer.

[34] M. Ramirez and H. Geffner. 2016. Heuristics for planning, plan recognition and parsing. *arXiv preprint arXiv:1605.05807* (2016).

[35] D. J. Rosenkrantz and R. E. Stearns. 1969. Properties of deterministic top down grammars. In *Proceedings of the 1st Annual ACM Symposium on Theory of Computing.* ACM, 165–180.

[36] R. W. Schvaneveldt. 1990. *Pathfinder Associative Networks: Studies in Knowledge Organization.* Ablex Publishing.

[37] A. Segal, S. Hindi, N. Prusak, O. Swidan, A. Livni, A. Palatnic, B. Schwarz, and Y. Gal. 2017. Keeping the teacher in the loop: Technologies for monitoring group learning in real-time. In *Proceedings of the International Conference on Artificial Inteligence in Education (AIED'17).*

[38] O. Seri and Y. Gal. 2014. Visualizing expert solutions in exploratory learning environments using plan recognition. In *Proceedings of the 19th International Conference on Intelligent User Interfaces.* ACM, 125–132.

[39] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artific. Intell.* 219 (2015), 40–66.

[40] S. Sohrabi, A. Riabov, and O. Udrea. 2016. Plan recognition as planning revisited. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence.* 3258–3264.

[41] T. C. Son, O. Sabuncu, C. Schulz-Hanke, T. Schaub, and W. Yeoh. 2016. Solving goal recognition design using ASP. In *Proceedings of the Conference of the Association for the Advancement of Artificial Intelligence (AAAI'16).* 3181–3187.

[42] G. Sukthankar, C. W. Geib, H. H. Bui, D. Pynadath, and R. P. Goldman. 2014. *Plan, Activity, and Intent Recognition: Theory and Practice.* Newnes.

[43] M. Vered, R. F. Pereira, M. C. Magnaguagno, G. A. Kaminka, and F. Meneguzzi. 2018. Towards online goal recognition combining goal mirroring and landmarks. AAMAS.

[44] C. Wayllace, P. Hou, and W. Yeoh. 2017. New metrics and algorithms for stochastic goal recognition design problems. In *Proceedings of the International Joint Conference of Artificial Intelligence (IJCAI'17)*.

[45] C. Wayllace, P. Hou, W. Yeoh, and T. C. Son. 2016. Goal recognition design with stochastic agent action outcomes. In *Proceedings of the International Joint Conference of Artificial Intelligence (IJCAI'16)*.

[46] S. Wiseman and S. Shieber. 2014. Discriminatively reranking abductive proofs for plan recognition. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS'14)*.

[47] 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.

Received February 2018; revised September 2018; accepted September 2018