Leveraging Imperfect Explanations for Plan Recognition Problems

Ahmad Alelaimat(✉), Aditya Ghose, and Hoa Khanh Dam

Decision Systems Lab, School of Computing and Information Technology, University of Wollongong, Wollongong, NSW 2522, Australia
{aama963,aditya,hoa}@uow.edu.au

Abstract. Open environments require dynamic execution of plans where agents must engage in settings that include, for example, re-planning, plan reusing, plan repair, etc. Hence, real-life Plan Recognition (PR) approaches are required to deal with different classes of observations (e.g., exogenous actions, switching between activities, and missing observations). Many approaches to PR consider these classes of observations, but none have dealt with them as deliberated events. Actually, using existing PR methods to explain such classes of observations may generate only so-called imperfect explanations (plans that partially explain a sequence of observations). Our overall approach is to leverage (in the sense of plan editing) imperfect explanations by exploiting new classes of observations. We use the notation of capabilities in the well-known Belief-Desire-Intention (BDI) agents programming as an ideal platform to discuss our work. To validate our approach, we show the implementation of our approach using practical examples from the Monroe Plan Corpus.

Keywords: Plan recognition · Plan updating · BDI agents

1 Introduction

It is generally accepted that an agent system with practical extensions can carry out tasks that would not otherwise be achievable by its basic reactive system. Often when the environment is highly dynamic and/or the task is complicated for the basic reactive behaviour of the agent system, developers resort to extending or adding new modules to the agent system. A typical example of extending agent models can be found far and wide in the state-of-the-art Belief-Desire-Intention (BDI) paradigm [1], including, but not limited to, extending the architecture with self-awareness [2], automated planning [3], and reconfigurability [4].

Much of the work done on PR involves abductive reasoning (e.g., [5,6], and [7]), which seeks to abductively infer plans by mapping the observed actions to plan libraries. A major drawback to Abductive Plan Recognition (APR) is that target

A. Ghose—Passed away prior to the submission of the manuscript. This is one of the last contributions by Aditya Ghose.

D. Calvaresi et al. (Eds.): EXTRAAMAS 2023, LNAI 14127, pp. 231–248, 2023.
https://doi.org/10.1007/978-3-031-40878-6_13

plans are usually not from the plan library. This can be due to several reasons, some of which are related to extending the observed agent system with different modules, such as the works presented in [2–4]. Actually, appealing to APR approaches to explain such observed actions would only generate imperfect explanations. An imperfect explanation is one that partially explains a sequence of actions. Another notion of PR is discovering plans by executing action models (in the sense of classical automated planning) to best explain the observed actions. Nevertheless, this can take a great deal of time in complex problems. Our approach is a third way which does not align itself with either notion. Still, it can potentially improve the explanatory power of plan libraries without the need for action model execution.

In this paper, we address the problem of *leveraging imperfect explanations*, the process of modifying existing hypotheses to explain an observed sequence of actions. We expect leveraging imperfect explanations to help answer the following questions:

- *What was the agent's original plan based on its new (or unusual) observed actions?*
- *Were the observed changes in a plan execution intentional?*

Answering these questions can be useful in understanding the capabilities of the observed agent. Consider, for example, a system composed of heterogeneous autonomous agents, some of which are equipped with a form of societal control (e.g., such as the one described in [8]). In such systems, leveraging imperfect explanations could be used to gain insight into the actual norm-modification operators of the observed agents compared to non-normative agents.

We show that when the observed agent operates in a domain model known to the observer, imperfect explanations can be a valuable guide to explain unknown plans that involve new classes of observations. Hence, our approach can be seen as a post-processing stage for various single-agent plan library-based PR techniques. To avoid arbitrary modification of hypotheses, we also introduce a classification model that can determine the settings (e.g., noisy or explanatory) in which an unknown plan has been observed. We demonstrate the performance of the proposed approach using the Monroe Plan Corpus.

The remainder of this work is organized as follows. Related work is discussed in Sect. 2. In Sect. 3, we introduce some preliminaries on the notion of capabilities and BDI agents programming, which are the two main ingredients of our work. Section 4 presents our running example with pointers to different scenarios. We formalize the problem of this work in Sect. 5. Section 6 describes our approach to leveraging imperfect explanations for PR problems. Empirical evaluation is described in Sect. 7 before we conclude and outline future work in Sect. 8.

2 Related Work

Real-life PR systems are required to deal with domains in which new classes of observations are frequently observed. Roughly speaking, there are three

noticeable domains regarding new classes of observations in PR problems: (1) Exploratory domains - the observed behaviour is a subject of exploratory and discovery learning (e.g., mistakes, exogenous and repeating activities), (2) Noisy domains - the observed behaviour is characterized by imperfect observability (e.g., extraneous, mislabeled and missing activities), and (3) Open domains - the observed behaviour is characterized by new, deliberated classes of observations (e.g., reconfigured plans). We concentrate our review on how existing PR approaches viewed/handled new classes of observations. We then use these classes in later sections where we describe our approach to leveraging imperfect explanations.

We first describe works that assume exploratory domains. With the intention of inferring students' plans, Mirsky et al. [7] proposed a heuristic plan recognition algorithm (called CRADLE) that incrementally prunes the set of possible explanations by reasoning about new observations and by updating plan arguments, in which explanations stay consistent with new observations. Uzan et al. [9] introduced an off-line PR algorithm (called PRISM) to recognize students' plans by traversing the plan tree in a way that is consistent with the temporal order of students' activities. Amir et al. [10] proposed an algorithm (called BUILDPLAN) based on recursive grammar to heuristically generate students' problem-solving strategies.

Many prior approaches to PR focused on dealing with noisy domains. Massardi et al. [11] classified noise in PR problems into three types: missing observations, mislabeled observations and extraneous actions and proposed a particle filter algorithm to provide robust-to-noise solutions to PR problems. Sohrabi et al. [12] transformed the PR problem into an AI planning problem that allows noisy and missing observations. Ramírez and Geffner [13] used classical planners to produce plans for a given goal G and compare these plans to the observed behaviour O. The probability distribution $\mathsf{P}(\mathsf{G}|\mathsf{O})$ can be computed by how the produced plans are close to the observed behaviour. Sukthankar and Sycara [14] proposed an approach for pruning and ranking hypotheses using temporal ordering constraints and agent resource dependencies.

PR systems are also required to deal with open domains, where the observed plans are usually not from the used plan library. Avrahami-Zilberbrand and Kaminka [15] describe two processes for anomalous and suspicious activity recognition: one using symbolic behaviour recognizer (SBR) and one using utility-based plan recognizer (UPR), respectively. Mainly, SBR filters inconsistent and ranking hypotheses, while UPR allows the observer to incorporate his preferences as a utility function. Zhuo et al. [16] also address the problem of PR in open domains using two approaches: one using an expectation-maximization algorithm and one using deep neural networks. A notable difference from other approaches is that the work of Zhuo et al. [16] is able to discover unobserved actions by constantly sampling actions and optimizing the probability of hypotheses.

Our work is not competing to, but complementing most of the previous works on PR, where it can be seen as a post-processing task for various single-agent plan library-based PR techniques. More precisely, leveraging imperfect explanations for PR problems can be viewed as an activity that occurs just before ruling out

imperfect explanations or considering an observation as exploratory or noisy. In contrast with the literature, we focus on improving the explanatory power of plan libraries by leveraging imperfect explanations and exploiting new classes of observations.

3 Preliminaries

This section reviews the prior research used in the remainder of this work. First, we clarify the link between the notion of capability and BDI programming, and then we describe the representation we use for capabilities and plans.

3.1 BDI Programming and Capabilities

An agent system with BDI architecture [1] commonly consists of a belief base (what the agent knows about the environment), a set of events (desires that the agent would like to bring about), a plan library (a set of predefined operational procedures), and a base of intentions (plans that the agent is committed to executing).

Fundamentally, the reactive behaviour of BDI agent systems includes the agent system handling events by selecting an event to address from the set of pending events, selecting a suitable plan from the plan library, and stacking its program into the intention base. A plan in the plan library is a rule of the form $\epsilon : \nu \leftarrow \varrho$, where the program ϱ is a predefined strategy to handle the event ϵ whenever the context condition ν is believed to be true by the agent. A plan can be selected for handling an event ϵ if it is relevant and applicable, i.e., designed with respect to the event ϵ, and the agent believes that the context of the plan ν is a logical consequence of its belief base, respectively. A program ϱ often can be presented as a set of actions that result in changes in the environment state. For the purpose of this work, we ignore other elements (e.g., trigger events or guards) in the plan body. Note that using BDI programming languages such as Jason [17] and 2APL [18], information on action pre- and post-conditions can only be defined in a separate file (called simulated environment), making this information invisible to the agent.

To reason about actions and their specifications, we need to access information about the preconditions and postconditions of all available actions. We shall refer to *capability* as an explicit specification of action preconditions and postconditions. A capability has been understood in intelligent agent studies as having at least one way to achieve some state of affairs, where it can be used only if its preconditions are believed to be true [2,19]. For the purposes of this work, we shall concentrate mostly on the plan library. We do not, therefore, discuss other issues related to integrating the notion of capabilities into the BDI paradigm. For a detailed introduction to integrating the notion of capabilities into the BDI system, the reader is referred to [2,19].

3.2 Capability and Plan Representation

Our representation of agent capabilities is closely related to [2,19], but significantly influenced by the action theory found in classical automated planning, such as what has been presented in situation calculus [20] and STRIPS reasoning [21]. Following this representation, we use a language of propositional logic $\mathcal{L}$ over a finite set of literals $\mathsf{L} = \{\mathsf{l}_1, \ldots, \mathsf{l}_n\}$ to represent the set of states of the environment S in which the agent is situated, such that each state of the environment $\mathsf{s} \in \mathsf{S}$ is a subset of L, i.e., $\mathsf{l}_i \in \mathsf{s}$ defines that the propositional literal l_i holds at the state s. As mentioned before, a capability specification describes an action that a BDI agent can carry out along with its pre- and post-conditions. Notationally, capability specification is triple subsets of L, which can be written as a rule of the form $\{\mathsf{pre(c)}\}\mathsf{c}\{\mathsf{post(c)}\}$, where

- $\{\mathsf{pre(c)}\}$ is a set of predicates whose satisfiability determines the applicability of the capability,
- c is the capability, and
- $\{\mathsf{post(c)}\}$ is a set of predicates that materialize with respect to the execution of the capability.

It is not hard to see that, by sequentially grouping capabilities that are dedicated to bringing about some state of affairs, the sequence $\mathsf{C} = \langle \mathsf{c}_1, \ldots, \mathsf{c}_n \rangle$ can be seen as an operational procedure to resolve that state. Consider the plan p, we use

- $\{\mathsf{pre(p)}\}$ as the conditions under which the plan is applicable,
- $\mathsf{C} = \langle \mathsf{c}_1, \ldots, \mathsf{c}_n \rangle$ as the plan body, and
- $\{\mathsf{post(p)}\}$ as the conditions associated with the end event of the plan,

As such, we use the notation $\mathsf{p} = \{\mathsf{pre(p)}\}\mathsf{C}\{\mathsf{post(p)}\}$ to represent plan p specifications.

For the purpose of reasoning about the execution of agent capabilities, we work with a simple representation of the possible ways the plan can be executed, which we term normative plan traces. A normative plan trace is one such that (1) the specification of capabilities completely determines the transition on states in S, i.e., if $\mathsf{s} \in \mathsf{S}$ and c is applicable to s, then it produces another state $\mathsf{s}' \in \mathsf{S}$, (2) the capabilities are guaranteed to execute sequentially, e.g., knowing that capability c_2 immediately follows capability c_1, then c_2 cannot be executed until $\mathsf{post}(\mathsf{c}_1)$ holds, and (3) a plan execution can not be interleaved with other plans.

4 Running Example

As a running example, we consider variants of Monroe County Corpus for emergency response domain [22].

Example 1. As shown in Fig. 1, the agent aims to provide medical attention to patients. It just receives requests from medical personnel, drives to the patient's

location (loc), loads the patient (pt) into the ambulance (amb), drives back to the hospital (h), and takes the patient out of the ambulance. Moreover, the agent is also equipped with capabilities related to the emergency response domain problem.

```
@h1 +provide_medical_attention(patient)
    :  available(ambulance)
    <- drive_to(ambulance,patient);
       get_in(patient,ambulance);
       drive_to(ambulance,hospital);
       get_out(patient,hospital).                              .
@h10 +!at(ambulance,loc)
    :  +at(ambulance,loc)
    <- true.
@h11 +!at(ambulance,loc)
    : not +at(ambulance,loc).
    <- drive_to(P);
    !at(ambulance,loc).
```

```
{not at(abm,loc)} drive_to(loc) {at(abm,loc)}
{at(abm,loc)} get_in(pt,amb) {in(pt,abm)}
{in(pt,abm),at(amb,h)} get_out(pt,amb) {out(pt,abm),at(pt,h)}
{not reachable(pt)} call(air_amb) {at(pt,h)}       .
{not breathing(pt),has(pt,pulse)} cpr(pt) {breathing(pt)}
```

Fig. 1. Providing medical attention plan and capabilities.

We argue that existing APR applications are inadequate in settings where the observed agent is characterized by extensibility and deliberation. To illustrate this, let us consider the following example.

Example 2. Consider the following scenarios that may arise in this emergency response domain:

i After arriving at the scene, the agent performed CardioPulmonary Resuscitation (CPR) on the patient and loaded the patient into the ambulance.
ii The agent called an air ambulance and drove back to the hospital without loading the patient into the ambulance due to rough terrain, poor weather, etc.
iii The agent drove back to the hospital without loading the patient into the ambulance as the patient went missing.

Although simple, Example 2 is far from trivial. First of all, it is not difficult to recognize that the basic reactive behaviour of the BDI agent system (described in the previous section) cannot produce the behaviours depicted in these scenarios on its own, since it does not have those plans in its plan library.

Arguably, there is at least an extension to the system that enabled such edits in the agent behaviour. Indeed, the state-of-the-art BDI agent framework exhibits a large number of extensions to the reactive behaviour of the BDI agent system. For example, the behaviour illustrated in scenario (i) involves an insertion edit (where the agent added CPR execution into the plan). It is possible for the agent to add an action(s) to a plan by incorporating automated planning into its system, such as in [3]. Scenario (ii) involves a substitution edit (where the agent replaced loading the patient into the ambulance with calling an air ambulance). It is possible for the agent to replace a capability with another one by leveraging an extension such as reconfigurability [4]. Finally, scenario (iii) represents a deletion edit (where the agent dropped loading and taking the patient into and out of the ambulance from the plan), which is doable if the agent is equipped with a task aborting mechanism, such as the one presented in [23].

5 Problem Formulation

A plan library H is a set of BDI plans, each of which contains a sequence of capabilities $\langle \mathsf{c}_1, \ldots, \mathsf{c}_n \rangle$ as its body, where each c_i, $1 \leq i \leq n$, is the capability name and a list of typed parameters. We assume the presence of a capability library, denoted by $\mathscr{A}$, comprises the set of all available capabilities specifications related to the domain problem. An observation of an *unknown plan* $\overline{\mathsf{p}}$ is denoted by $\mathsf{O} = \langle \mathsf{o}_1, \ldots, \mathsf{o}_n \rangle$, where $\mathsf{o}_i \in \mathscr{A} \cup \{\varnothing\}$, i.e., the observation o_i is either a capability in $\mathscr{A}$ or an empty capability $\varnothing$ that has not been observed. Note that the plan $\overline{\mathsf{p}}$ is not necessarily in H, and thus mapping O to the H may generate only imperfect explanations.

When reasoning about new classes of observation, one can classify an observed capability by four types: (1) match, when the capability is correctly observed, (2) insertion edit, when the observed capability is added to a normative plan trace, (3) deletion edit, when the observed capability is dropped from a normative plan trace, (4) substitution edit, when a capability that is to be executed as part of a normative plan trace is replaced with another one. We propose to describe these three edits in plan execution using operations as follows.

Definition 1 (Abductive edit operation, sequence). Let p with $\mathsf{C} = \langle \mathsf{c}_1, \ldots, \mathsf{c}_n \rangle$ as its body be an imperfect explanation for the observation $\mathsf{O} = \langle \mathsf{o}_1, \ldots, \mathsf{o}_n \rangle$. An abductive edit operation is the insertion, deletion, or substitution of capabilities in C according to the observations in O. An abductive insertion of an observed capability o_i is denoted by $(\varnothing \rightarrow \mathsf{o}_i)$, deletion of c_i is denoted by $(\mathsf{c}_i \rightarrow \varnothing)$ and substitution of c_i with o_i is denoted by $(\mathsf{c}_i \rightarrow \mathsf{o}_i)$. An abductive edit sequence $\mathsf{AES} = \langle \mathsf{ae}_1, \ldots, \mathsf{ae}_n \rangle$ is a sequence of abductive edit operations. An AES derivation from C to O is a sequence of sequences $\mathsf{C}_0, \ldots, \mathsf{C}_n$, such that $\mathsf{C}_0 = \mathsf{C}$ and $\mathsf{C}_n = \mathsf{O}$ and for all $1 \leqslant i \leqslant n$, $\mathsf{C}_{i-1} \rightarrow \mathsf{C}_i$ via ae_i.

Definition 2 (Extended plan library). An Extended plan library is a couple $\mathsf{EPL} = (\mathsf{H}, \mathsf{AES})$, where

1. H is a plan library, and

2. AES is a sequence of abductive edit operations.

Definition 3 (APR problem). Considering our settings, the APR problem can be defined by a 4-tuple APR = (EPL, O, explain, $\mathscr{A}$), where:

1. EPL is an extended plan library,
2. O is an observed trace of capabilities,
3. explain is a map from plans and sub-plans of H to subset of O, and
4. $\mathscr{A}$ is a library of capability specifications.

As such, the solution to APR is to discover an unknown plan $\overline{p}$, which is a plan with an edited sequence of capabilities as its body that best explains O given EPL and $\mathscr{A}$. Again, this can be challenging since the plan $\overline{p}$ is not necessarily in H, and thus mapping O to the H may generate only imperfect explanations.

6 Approach

Our approach to leveraging imperfect explanations consists of four phases, as shown in Fig. 2. These phases are (1) Classification of unknown plans, (2) Abductive editing of imperfect explanations, (3) Reasoning about the validity of the edited plan, and (4) Abductive updating of imperfect explanations. We describe each of these steps in greater detail in the following sub-sections. Note that solid arrows refer to leveraging imperfect explanations phases and dotted arrows refer to required inputs.

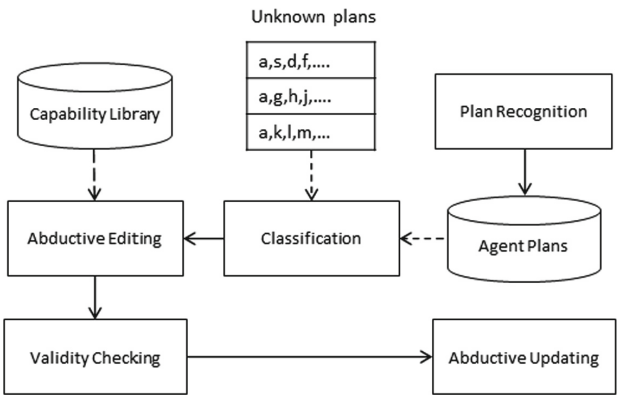

Fig. 2. Overview of the approach.

As illustrated in Fig. 2, leveraging imperfect explanations takes as inputs

1. An unknown plan,
2. An approximation of the plan(s) that have been used to generate input (1), and

3. A set of all available capabilities specifications related to the domain problem.

While inputs (1) and (2) are used in the classification and updating phases, inputs (2) and (3) are used in the editing and validation phases. Note also that our approach still requires an external plan recognition mechanism to provide imperfect explanations (hence the dotted arrow from agent plans to the classification phase).

6.1 Classification

Given an unknown plan, before any decision can be made concerning leveraging imperfect explanations, it is first necessary to determine the characteristic of the environment in which the unknown plan has been carried out (i.e., we do not want to build on noisy or exploratory observations). To that end, we use decision tree learning to classify unknown plans. Although decision trees are not the only means of classification, they are highly interpretable models [24]. Following the classification described in Sect. 2, the following taxonomy for classification is proposed:

EE Exploratory environment - observed behaviour is a subject of exploratory or discovery learning. Much of the work done on PR for exploratory domains considers trial-and-error, activity repeating and interleaving as features of exploratory behaviours [7,9,10].

NE Noisy environment - observed behaviour is characterized by imperfect observability. Previous studies (e.g., [12] and [11]) reported noisy observations as those that cannot be explained by the actions of any plan for any given goal (computing all possible plans for a given goal is fully described in [6]).

OE Open environment - observed behaviour is characterized by extensibility. Many studies on intelligent agents (e.g., [4,25,26]) consider rational changes in plan execution (see Sect. 6.3 for how rational changes are validated) as a feature of engaging the agent with open environments.

A given unknown plan is classified into one of the classes {**EE, NE, OE**}, each representing the settings in which the unknown plan has been executed. An unknown plan is assigned to a class membership based on its characteristic features by comparing it to its imperfect explanation (i.e., approximation of the unknown plan). For example, unknown plans that contain actions that any plan for any given goal cannot explain are labelled as **NE**, whilst unknown plans that contain rational edits compared to their imperfect explanations are labelled as **OE**. Historical instances are labelled manually while the test data is not labelled, so the decision tree can classify whether the unknown plan is a result of **EE**, **NE**, or **OE**.

According to state-of-the-art PR and intelligent agents [4,6,7,9–12,25,26], we initially extracted a number of features related to **EE**, **NE**, and **OE**.

1. **Unreliable action**: This binary feature represents whether an unknown plan contains action(s) that any plan for any given goal cannot explain.

2. **Trial-and-error**: This binary feature represents whether an unknown plan contains multiple attempts to achieve a desirable effect using different activities.
3. **Action repeating**: This binary feature represents whether an unknown plan contains multiple attempts to achieve a desirable outcome using the same activity with different parameters.
4. **Activity interleaving**: This binary feature represents whether an unknown plan contains the execution of an activity while waiting for the results of the current activity.
5. **Rational editing**: This binary feature represents whether an unknown plan contains rational edit(s) compared to its normative plan traces.

Classification takes place as a supervised multi-class classification making use of the features described above. Classification of unknown plans is applied before the actual process of leveraging imperfect explanations due to avoid useless wait (i.e., we do not want to build on noisy or exploratory observations). If an unknown plan is classified as **OE**, the plan will be taken as input. Continuing with our running example, both scenarios (i) and (ii) were classified as **OE**. This is due to containing the feature of rational changes, which strongly correlates to **OE**. Whiles scenario (iii) was classified as a **NE** because any plan of the given goal could not explain it.

In this section, we have seen one possible way of classifying changes in plan execution by attending to the settings in which it has been carried out (i.e., noisy, explanatory and open). However, in fact, many other features can be adopted to classify these changes hence a better understanding of what the target agent is actually doing and why. For example, these changes can be divided into mistakes and exploratory activities in exploratory domains. Another example, noisy observations can be classified as sensor failure or programming errors (e.g., inappropriate belief revision). However, such classifications are outside the scope of the present work, since this is, in general, an intractable problem.

6.2 Abductive Editing

Our guiding intuition here is that a plan that serves as an imperfect explanation for an observed behaviour could possibly be edited (modified) according to that observation, thus improving the explanatory power of the plan library. We realize a computational solution to leveraging imperfect explanations by appealing to the optimal edit distance [27] between an unknown plan and its imperfect explanation and using its corresponding edit sequence.

Let plan p with body $\mathsf{C} = \langle \mathsf{c}_1, \ldots, \mathsf{c}_\mathsf{n} \rangle$ be an imperfect explanation of the observations $\mathsf{O} = \langle \mathsf{o}_1, \ldots, \mathsf{o}_\mathsf{m} \rangle$, with the former having length n and the latter length m. Recall that turning the plan body C into O requires a sequence of edit operations. Each of these operations can be weighted by a cost function, denoted by $\mathsf{w}(\mathsf{ae})$. For example, one can set the cost function to return 0 when the capability is correctly observed and to return 1 otherwise.

With a cost function in hand, the abductive plan edit distance between C and O is given by a matrix d of size $\mathsf{n} \times \mathsf{m}$, defined by the recurrence.

$$d[i,0] = i$$
$$d[0,j] = j$$

$$d[i,j] = \begin{cases} d[i-1,j-1] & \text{(correctly observed)} \\ \min \begin{cases} d[i-1,j] + w(\varnothing, o_i) & \text{(insertion edit)} \\ d[i,j-1] + w(c_i, \varnothing) & \text{(deletion edit)} \\ d[i-1,j-1] + w(c_i, o_i) & \text{(substitution edit)} \end{cases} \end{cases}$$

After filling the matrix, the value in the bottom-right cell of the d, or $d[m,n]$, will represent the minimum cost to turn the plan body C into the observation sequence O, and this cost is the abductive plan edit distance. For the corresponding AES to be computed, we need to traceback the choices that led to the minimum edit cost in the above recurrence. Hence, turning the imperfect explanation p into an unknown plan $\overline{p}$ that best explains the observations in O can be seen as applying an AES that corresponds to the abductive plan edit distance of p.

Example 3. Continuing with our running example, assume the imperfect explanations h1 with a body as shown below:

```
drive_to(loc),get_in(pt,amb),drive_to(loc),get_out(pt,amb).
```

An observation sequence O from scenario (ii) as shown below:

```
drive_to(pt),call(air_amb),drive_to(h)
```

And let $w(ae) = 0$ when the capability is correctly observed and $w(ae) = 1$ otherwise. Based on the above inputs, the abductive edit operations needed to turn the body part of plan h1 into O are:

1. $\text{get\_in}(pt, amb) \rightarrow \text{call}(\text{air\_amb})$, and
2. $\text{get\_out}(pt, amb) \rightarrow \varnothing$.

Example 3 illustrates how imperfect explanations can be leveraged merely by editing. Although edits in Example 3 sound rational, they may be invalid in other scenarios. Hence, there are two important questions yet to be discussed: how to ensure (1) that the edited plan is consistent and (2) at the end of its execution, the goal is achieved. We will address these two questions in the following subsection.

6.3 Validity Checking

We build on the approach of monitoring plan validity proposed by [28]. Our theory of edited plan validity uses the accumulative effects (denoted as accum) of [29] and ensures consistency during the process of plan editing, given the capability library $\mathscr{A}$. To monitor an edited plan validity, two plans are generated

for every PR problem - one with the minimum edit cost, i.e., the imperfect explanation, and one that explains O completely, i.e., $\mathsf{explain}(\overline{\mathsf{p}}) = \mathsf{O}$. Assuming an idealised execution environment, plan validity can be defined as follows.

Definition 4 (Valid Plan). Consider the capability library $\mathscr{A}$ and a plan specification $\mathsf{p} = \{\mathsf{pre(p)}\}\mathsf{C}\{\mathsf{post(p)}\}$ for plan p with body $\mathsf{C} = \langle \mathsf{c}_1, \ldots, \mathsf{c}_n \rangle$, we say that the plan p is valid in the state s if

1. $\mathscr{A} \models (\langle \mathsf{pre}(\mathsf{c}_1), \ldots, \mathsf{pre}(\mathsf{c}_n) \rangle, \mathsf{s})$, and
2. $\mathsf{accum(p)} \models \mathsf{post(p)}$.

As such, with respect to $\mathscr{A}$ and the current state of the world s, the preconditions of each capability in the plan body of p will be satisfied, and the final effect scenario $\mathsf{accum(p)}$ associated with the end state event of the plan p execution entails its post-condition specifications. An important detail of this definition is that a single edit can impact the consistency of the plan. Also, it can change the final effect scenario associated with the end state of the plan. We now identify a valid edited plan.

Definition 5 (Valid edited plan). Consider the plan $\mathsf{p} = \{\mathsf{pre(p)}\}\mathsf{C}\{\mathsf{post(p)}\}$ as an imperfect explanation of O. Let $\overline{\mathsf{p}}$ be an edited plan that has identical pre- and post-conditions to p except that $\overline{\mathsf{p}}$ has an edited sequence of capabilities $\overline{\mathsf{C}}$. We say that the edited plan $\overline{\mathsf{p}}$ preserves the validity of p if $\overline{\mathsf{p}} = \{\mathsf{pre(p)}\}\overline{\mathsf{C}}\{\mathsf{post(p)}\}$ is valid.

Where it is possible for the plan recognition model to find two or more different imperfect explanations that have the same edit cost and are valid, further post-processing may be required for more reliable results. We overtake this problem by seeking stronger goal entailment and consistency conditions, such as the plan internal analysis described in [30].

6.4 Abductive Updating

Now, we consider the problem of what needs to be done to improve the explanatory power of the plan library when an edited plan is found valid according to the checks described above. An easy solution is to create a new plan that has identical triggering and context parts to the imperfect explanation, except that $\overline{\mathsf{p}}$ has an edited body $\overline{\mathsf{C}}$ that explains O, i.e., $\mathsf{explain}(\overline{\mathsf{p}}) = \mathsf{O}$. However, this may increase the complexity of determining applicable plans at run-time. More interestingly, we offer a semi-automated solution for merging an edited plan with its imperfect explanation.

Procedure 1 (Abductive Updating). Consider the imperfect explanation p and its edited plan $\overline{\mathsf{p}}$ and let $\mathsf{AES} = \langle \mathsf{ae}_1, \ldots, \mathsf{ae}_n \rangle$ be a derivation from p to $\overline{\mathsf{p}}$.

```
1: For each ae_i in AES:
2: Replace c_i with subgoal sg_i
3: Create two subplans p1 and p2, and let
4: triggering(p1) = triggering(p2) = sg_i
5: switch(ae_i):
6:   case(c_i → o_i)
7:     context(p1) = pre(c_i)
8:     context(p2) = pre(o_i)
9:     body(p1) = c_i
10:    body(p2) = o_i
11:  case(c_i → ∅)
12:    context(p1) = pre(c_i)
13:    context(p2) = not pre(c_i)
14:    body(p1) = c_i
15:    body(p2) = true
16:  case(∅ → o_i)
17:    context(p1) = not pre(o_i)
18:    context(p2) = pre(o_i)
19:    body(p1) = true
20:    body(p1) = o_i
```

Procedure 1 facilitates the merging between an imperfect explanation, i.e., one with the minimum edit cost, and its edited plan, i.e., one that explains O completely. Fundamentally, for each ae_i in AES, the procedure replaces the corresponding capability with a sub-goal sg_i (line 2), for which two sub-plans p_1 and p_2 are created (line 3–4). Our guiding intuition behind creating two sub-plans is to improve the explanatory power of the plan library while maintaining its construction.

Recall that there are three ways in which a plan can be edited: deletion, insertion, and substitution. Hence, there are three ways in which sub-plans can be created (line 5). For example, let us consider the substitution edit (line 6): In this case, plan p_1 takes the corresponding capability c_i as its body and $\mathsf{pre}(\mathsf{c}_i)$ as its context part (lines 7 and 9). While p_1 takes the corresponding observation o_i as its body and $\mathsf{pre}(\mathsf{o}_i)$ as its context part (lines 8 and 10). However, there are situations where valid edited plans could possibly be merged with their imperfect explanations, but they should not be merged. For example, avoiding negative interactions between goals. For readers interested in how we can deal with the feasibility of plans merging, we refer to [31].

7 Evaluation

In this section, we present the evaluation of our approach. First, we present the setup for evaluation. Next, evaluation results are described.

7.1 Evaluation Setup

We implemented our approach as a plugin for our Toolkit XPlaM[1] [32] and evaluated it using Monroe Plan Corpus [22]. For the purposes of this work, the corpus has been rewritten using AgentSpeak(L) programming language and executed using Jason interpreter [17]. Observation traces[2] have been collected using a debugging tool called Mind Inspector [17]. Unknown plans have been classified using C4.5 decision tree algorithms [33]. Abductive plan editing is implemented using the well-known Levenshtein distance. For deriving accumulative effects, we implemented a state update operator similar to the one described in [29] using the NanoByte SAT Solver package[3]. For testing, experiments have been implemented on an Intel Core i5-6200 CPU (with 8 GB RAM).

7.2 Performance Results

We ran a number of experiments to test the scalability of our approach with respect to the number of new classes of observations. The corpus used in these experiments has been executed to consider all possible traces (the number of executed actions is 80). New classes of observations were artificially added to the observation traces. The number of capabilities in $\mathscr{A}$ is set to 20. Figure 3 compares the number of explanations generated by our approach for a different number of new classes of observations.

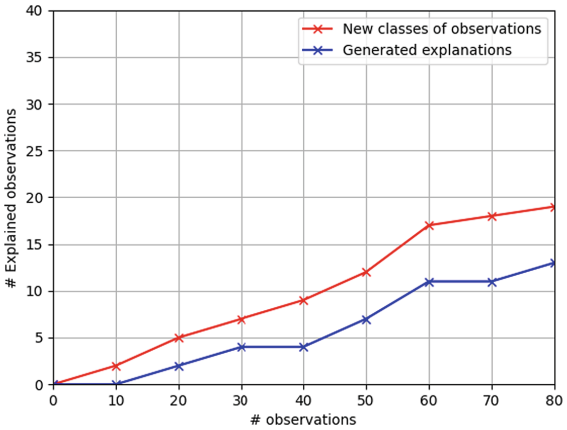

Fig. 3. Performance results.

[1] The source code for XPlaM Toolkit (including the code for the approach presented here) has been published online at https://github.com/dsl-uow/xplam.

[2] We published the datasets supporting the conclusions of this work online at https://www.kaggle.com/datasets/alelaimat/xplam.

[3] https://github.com/nano-byte/sat-solver.

Our first study shows that with a moderate number of new classes of observations and a reasonable number of capabilities in $\mathscr{A}$, abductive plan editing would possibly improve the explanatory power of plan libraries in open environment settings. Note that the scalability of the abductive plan editing should be tied to the performance of the classification task. Nevertheless, the results depicted in Fig. 3 are partially independent of the used classifier (C4.5 decision tree). Actually, we allowed for unknown plans that contain unreliable actions to be considered as inputs. We argue that additional features are required for a more accurate classification of unknown plans.

7.3 Speed

Aiming to find out how fast abductive plan editing would take to explain unknown plans, we ran a number of experiments with Monroe Plan Corpus. Recall that executing action models (i.e., planning) is another technique to explain unknown plans. Hence, we use diverse planning [34] as a benchmark to assess the complexity of our approach. Diverse planning aims at discovering a set of plans that are within a certain distance from each other. The discovered set is then used to compute the closest plan to the observation sequence. Figure 4 shows the time required to discover a valid explanation for variate numbers of new classes of observations using diverse planning (as used by LPG-d planner [34]) to the time required by our plan editing approach.

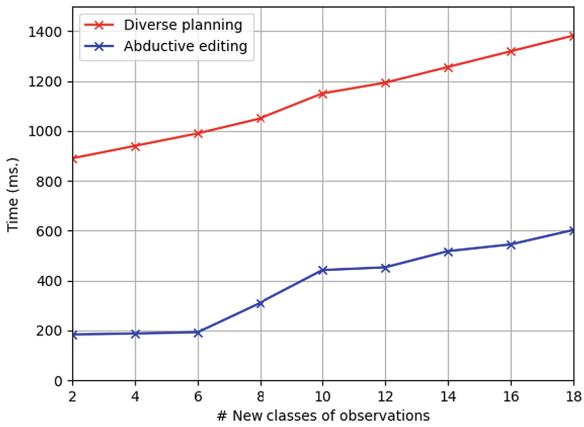

Fig. 4. Explaining time required for plan editing and diverse planning.

Figure 4 shows that, unlike diverse planning, for different number of new classes of observations, plan editing is a relatively faster approach to explaining unknown plans. However, the reader should keep two details in mind. First, the performance of plan editing is tied to the performance of the used dynamic programming algorithm (in our case, Levenshtein distance). A more fine-grained

evaluation, therefore, should include different dynamic programming algorithms (e.g., Hamming distance). Secondly, during our experiments, we noticed that the size of $\mathscr{A}$ can highly affect the performance of diverse planning and plan editing. We will investigate these two details as part of our future work.

8 Conclusion

Much of the work done on APR requires a plan library to infer the top-level plans of the observed agent. Nevertheless, in open environment settings, target plans are usually not from plan libraries due to reusing plans, replanning and agent self-awareness, etc. This work builds on a more sophisticated notion of APR, which seeks to improve the explanatory power of plan libraries by way of leveraging imperfect explanations and exploiting new classes of observations. In this work, we proposed a classification of unknown plans based on the characteristics of the environment in which they have been carried out. As far as we know, this has been absent in PR research. Furthermore, we presented a theory based on capabilities and plans and introduced the notion of abductive plan editing. Finally, we described how imperfect explanations could be updated with new classes of observations in a rational fashion.

A number of extensions of this work are of interest, including applications of plan library reconfigurability [4], plan editing in online settings, and dealing with incomplete action models (i.e., learning unknown activities). Furthermore, we plan to improve our approach in order to deal with logs obtained from noisy and exploratory domains and compare its performance with state-of-the-art plan recognition methods, where incompleteness of knowledge and non-determinism might be present.

References

1. Georgeff, M., Rao, A.: Modeling rational agents within a BDI-architecture. In: Proceedings of the 2nd International Conference on Knowledge Representation and Reasoning, KR 1991, pp. 473–484. Morgan Kaufmann (1991)
2. Padgham, L., Lambrix, P.: Agent capabilities: extending BDI theory. In: AAAI/IAAI, pp. 68–73 (2000)
3. De Silva, L., Sardina, S., Padgham, L.: First principles planning in BDI systems. In: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, vol. 2, pp. 1105–1112 (2009)
4. Cardoso, R.C., Dennis, L.A., Fisher, M.: Plan library reconfigurability in BDI agents. In: Dennis, L.A., Bordini, R.H., Lespérance, Y. (eds.) EMAS 2019. LNCS (LNAI), vol. 12058, pp. 195–212. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51417-4_10
5. Singla, P., Mooney, R.J.: Abductive Markov logic for plan recognition. In: 25th AAAI Conference on Artificial Intelligence (2011)
6. Geib, C.W., Goldman, R.P.: A probabilistic plan recognition algorithm based on plan tree grammars. Artif. Intell. **173**(11), 1101–1132 (2009)

7. Mirsky, R., Gal, Y., Shieber, S.M.: CRADLE: an online plan recognition algorithm for exploratory domains. ACM Trans. Intell. Syst. Technol. (TIST) **8**(3), 1–22 (2017)
8. Meneguzzi, F.R., Luck, M.: Norm-based behaviour modification in BDI agents. In: AAMAS, vol. 1, pp. 177–184 (2009)
9. Uzan, O., Dekel, R., Seri, O., et al.: Plan recognition for exploratory learning environments using interleaved temporal search. AI Mag. **36**(2), 10–21 (2015)
10. Amir, O., et al.: Plan recognition in virtual laboratories. In: 22nd International Joint Conference on Artificial Intelligence (2011)
11. Massardi, J., Gravel, M., Beaudry, E.: Error-tolerant anytime approach to plan recognition using a particle filter. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 29, pp. 284–291 (2019)
12. Sohrabi, S., Riabov, A.V., Udrea, O.: Plan recognition as planning revisited. In: IJCAI, pp. 3258–3264 (2016)
13. Ramírez, M., Geffner, H.: Probabilistic plan recognition using off-the-shelf classical planners. In: 24th AAAI Conference on Artificial Intelligence (2010)
14. Sukthankar, G., Sycara, K.P.: Hypothesis pruning and ranking for large plan recognition problems. In: AAAI, vol. 8, pp. 998–1003 (2008)
15. Avrahami-Zilberbrand, D., Kaminka, G.A.: Keyhole adversarial plan recognition for recognition of suspicious and anomalous behavior. In: Plan, Activity, and Intent Recognition, pp. 87–121 (2014)
16. Zhuo, H.H., Zha, Y., Kambhampati, S., Tian, X.: Discovering underlying plans based on shallow models. ACM Trans. Intell. Syst. Technol. (TIST) **11**(2), 1–30 (2020)
17. Bordini, R.H., Fred Hübner, J., Wooldridge, M.: Programming Multi-agent Systems in AgentSpeak Using Jason, vol. 8. Wiley (2007)
18. Dastani, M.: 2APL: a practical agent programming language. Auton. Agent. Multi-Agent Syst. **16**, 214–248 (2008)
19. Padgham, L., Lambrix, P.: Formalisations of capabilities for BDI-agents. Auton. Agent. Multi-Agent Syst. **10**(3), 249–271 (2005). https://doi.org/10.1007/s10458-004-4345-2
20. Reiter, R.: The frame problem in the situation calculus: a simple solution (sometimes) and a completeness result for goal regression. In: Artificial and Mathematical Theory of Computation, pp. 359–380. Citeseer (1991)
21. Fikes, R.E., Nilsson, N.J.: Strips: a new approach to the application of theorem proving to problem solving. Artif. Intell. **2**(3–4), 189–208 (1971)
22. Blaylock, N., Allen, J.: Generating artificial corpora for plan recognition. In: Ardissono, L., Brna, P., Mitrovic, A. (eds.) UM 2005. LNCS (LNAI), vol. 3538, pp. 179–188. Springer, Heidelberg (2005). https://doi.org/10.1007/11527886_24
23. Thangarajah, J., Harland, J., Morley, D., Yorke-Smith, N.: Aborting tasks in BDI agents. In: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1–8 (2007)
24. Gilpin, L.H., Bau, D., Yuan, B.Z., Bajwa, A., Specter, M., Kagal, L.: Explaining explanations: an overview of interpretability of machine learning. In: 2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA), pp. 80–89. IEEE (2018)
25. Stringer, P., Cardoso, R.C., Huang, X., Dennis, L.A.: Adaptable and verifiable BDI reasoning. arXiv preprint arXiv:2007.11743 (2020)
26. Dennis, L.A., Fisher, M.: Verifiable self-aware agent-based autonomous systems. Proc. IEEE **108**(7), 1011–1026 (2020)

27. Navarro, G.: A guided tour to approximate string matching. ACM Comput. Surv. (CSUR) **33**(1), 31–88 (2001)
28. Fritz, C., McIlraith, S.A.: Monitoring plan optimality during execution. In: ICAPS, pp. 144–151 (2007)
29. Ghose, A., Koliadis, G.: Auditing business process compliance. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 169–180. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74974-5_14
30. Gou, Y.: A computational framework for behaviour adaptation: the case for agents and business processes (2018)
31. Thangarajah, J., Padgham, L., Winikoff, M.: Detecting & exploiting positive goal interaction in intelligent agents. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 401–408 (2003)
32. Alelaimat, A., Ghose, A., Dam, H.K.: XPlaM: a toolkit for automating the acquisition of BDI agent-based digital twins of organizations. Comput. Ind. **145**, 103805 (2023)
33. Quinlan, J.R.: C4.5: Programs for Machine Learning. Elsevier (2014)
34. Nguyen, T.A., Do, M., Gerevini, A.E., Serina, I., Srivastava, B., Kambhampati, S.: Generating diverse plans to handle unknown and partially known user preferences. Artif. Intell. **190**, 1–31 (2012)