Using Sub-Optimal Plan Detection to Identify Commitment Abandonment in Discrete Environments

RAMON FRAGA PEREIRA, PUCRS
NIR OREN, University of Aberdeen
FELIPE MENEGUZZI, PUCRS

Assessing whether an agent has abandoned a goal or is actively pursuing it is important when multiple agents are trying to achieve joint goals, or when agents commit to achieving goals for each other. Making such a determination for a single goal by observing only plan traces is not trivial, as agents often deviate from optimal plans for various reasons, including the pursuit of multiple goals or the inability to act optimally. In this article, we develop an approach based on domain independent heuristics from automated planning, landmarks, and fact partitions to identify sub-optimal action steps—with respect to a plan—within a fully observable plan execution trace. Such capability is very important in domains where multiple agents cooperate and delegate tasks among themselves, such as through *social commitments*, and need to ensure that a delegating agent can infer whether or not another agent is actually progressing towards a delegated task. We demonstrate how a creditor can use our technique to determine—by observing a trace—whether a debtor is honouring a commitment. We empirically show, for a number of representative domains, that our approach infers sub-optimal action steps with very high accuracy and detects commitment abandonment in nearly all cases.

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

Additional Key Words and Phrases: Commitments, plan abandonment, plan execution, landmarks, domain-independent heuristics, optimal plan, sub-optimal plan

ACM Reference format:
Ramon Fraga Pereira, Nir Oren, and Felipe Meneguzzi. 2020. Using Sub-Optimal Plan Detection to Identify Commitment Abandonment in Discrete Environments. *ACM Trans. Intell. Syst. Technol.* 11, 2, Article 23 (January 2020), 26 pages.
https://doi.org/10.1145/3372119

Preliminary versions of parts of this article appeared as a two-page extended abstract [29] and a workshop paper [31]. This article expands the problem formulation and formalisation, the descriptions and discussion of the heuristics and their implications for the technique, working examples and explanations, and experimentation.
This work was financed by the Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior (Brazil, Finance Code 001). F. Meneguzzi thanks CNPq for partial financial support under its PQ fellowship, grant number 305969/2016-1.
Authors' addresses: R. F. Pereira and F. Meneguzzi, School of Technology, PUCRS, Porto Alegre, Brazil; emails: ramon.pereira@edu.pucrs.br, felipe.meneguzzi@pucrs.br; N. Oren, Department of Computing Science, University of Aberdeen, Aberdeen, Scotland; email: n.oren@abdn.ac.uk.

2157-6904/2020/01-ART23 $15.00
https://doi.org/10.1145/3372119

1 INTRODUCTION

Autonomous agents generate and execute plans in pursuit of goals. Rationality would require such agents to execute plans which are—in some sense—*optimal*. However, an agent may execute additional actions that are not part of an optimal plan due to factors including indecision, an imperfect planning mechanism, interleaving concurrent plans for multiple goals, and, in the most extreme case, goal or plan abandonment. Sub-optimal execution of a plan or abandonment of a goal is not a major problem for an individual agent acting on its own; however, when agents work together and delegate goal achievement to each other, they need to be able to monitor and detect when an agent committed to acting on its behalf fails to comply with such commitment. Thus, determining whether observed actions are sub-optimal is often important, especially when goal delegation has taken place; where one agent is obliged or committed to achieve a goal; or where agents are coordinating plan execution in pursuit of a joint goal. In all of these cases, determining that an agent is acting sub-optimally allows other agents to re-plan, notify or warn the agent, apply sanctions, or otherwise mitigate against the effects of the failure to achieve a certain state of affairs in a timely fashion.

Consider, for example, a situation where agents (e.g., trucks and airplanes) work together to deliver items to various destinations. By making commitments to each other, they are able to come up with a joint plan to perform item delivery. Clearly, all agents are interested in monitoring each other for deviations, enabling them to replan or impose sanctions if one of their partners begins behaving in an unexpected manner. Identifying such deviations involves monitoring how the partners are executing their part of the plan and, from this, determining if they are still committed to achieving it. Although one could determine whether it is no longer possible for a partner to achieve the joint plan (e.g., if a truck can no longer deliver an item on time), it is useful to detect deviations earlier (e.g., detecting if the truck is moving away rather than towards its destination).

In this work, we address this problem of monitoring plan execution by detecting which steps in a plan are sub-optimal—that is, not contributing towards the agent's goal. By using a threshold on the number of sub-optimal actions, we can decide whether an agent has abandoned a monitored goal. The use of the threshold value gives some flexibility to the observed agent to execute/perform actions that are part of other plans (e.g., consider that the agent has other goals to achieve). Our contribution is twofold: first, we develop efficient techniques to compute whether a plan is sub-optimal and which actions in this plan are sub-optimal; second, we leverage this technique to identify whether an agent is individually committed to achieve a particular goal, allowing us to identify whether this agent will honour a social commitment [41].

The techniques we develop exploit domain-independent heuristics [12], planning landmarks [15], and fact partitions [26] to monitor plan optimality and goal achievability (Section 3). We assume that during plan execution, all actions performed by an agent are visible, and that a monitored goal and a domain theory (in a planning language) are available. We then evaluate the optimality of plan steps (i.e., actions) in two ways. First, we estimate a distance (using any domain-independent heuristic) to the monitored goal at each step, analysing possible deviations. Second, we evaluate how each observation contributes towards a goal by analysing how they diminish their estimated distance to a sequence of states that must be achieved for the goal to eventually be achieved, also referred to as *landmarks* [15]. With this information, we can infer which observed actions are (probably) not part of an optimal plan. Our optimality monitoring approach can be contrasted with previous work on detecting whether a plan being executed aims to a monitored goal [9]. The work of Fritz and McIlraith [9] relies on a complex logical formalism and focused on extraneous events rather than directly on an agent's behaviour. We formalise the problem of *commitment abandonment* detecting and the relation of an individual commitment to a plan in Section 4, using our plan optimality monitoring approach to detect whether an agent has

abandoned a social commitment. This allows the creditor (observer) to ascertain at runtime whether, and when, the debtor fails to honour the commitment at the agreed upon quality.

Experiments over several planning domains (Section 5) show that our approaches yield high accuracy at low computational cost to detect sub-optimal actions (i.e., which actions do not contribute to achieve a monitored goal) and can, in nearly all evaluated cases, detect whether a debtor agent has abandoned a commitment.

2 BACKGROUND

In this section, we review essential background on automated planning terminology, domain-independent heuristics, landmarks, and fact partitioning.

2.1 Planning

Planning is the problem of finding a sequence of actions (i.e., a plan) that achieves a particular goal from an initial state. We adopt the terminology of Ghallab et al. [12] to represent planning domains and instances (also called *planning problems*) in Definitions 1 through 5. We begin by considering *states*, built up of predicates, which describe the environment at a moment in time.

Definition 1 (Predicates and State). A predicate is denoted by an n-ary predicate symbol p applied to a sequence of zero or more terms $(\tau_1, \tau_2, \ldots, \tau_n)$—terms are either constants or variables. We denote the set of all possible predicates as Ψ. We denote Σ as the set of facts, which comprise all grounded predicates in both their positive or negated forms, as well as constants for truth ($\top$) and falsehood ($\bot$). A state is a finite set of positive grounded predicates, representing logical values that are true in the state.

Planning domains describe the environment's properties and dynamics through operators, which use a first-order representation to define schemata for state-modification actions according to Definitions 2 through 5.

Definition 2 (Operators and Actions). An operator a is a triple $\langle name(a), pre(a), eff(a)\rangle$, where $name(a)$ is the description or *signature* of a, $pre(a)$ are its preconditions—the set of predicates that must exist in the current state for a to be executed, and $eff(a)$ represents the effects of a which modify the current state. Effects are split into $eff(a)^+$ (i.e., an add-list of positive predicates) and $eff(a)^-$ (i.e., a delete-list of negated predicates). An action is then a grounded operator instantiated over its free variables. The set of all operators is denoted by $\mathcal{O}$, and the set of all possible actions is $\mathcal{A}$.

We say an action a is applicable to a state S if and only if $S \models pre(a)$. This action generates a new state $S' := (S \cup eff(a)^+)/eff(a)^-$. The application of an applicable action is captured through a function $\gamma(S, A)$ as follows:

$$\gamma(S, A) = \begin{cases} (S \cup eff(a)^+)/eff(a)^- & \text{if } S \models pre(a) \\ \bot & \text{otherwise.} \end{cases}$$

Definition 3 (Planning Domain). A planning domain definition Ξ is represented by a pair $\langle \Sigma, \mathcal{A}\rangle$, and consists of a finite set of grounded facts Σ (e.g., environment properties) and a finite set of grounded actions $\mathcal{A}$.

A *planning instance* comprises both a *planning domain* and the elements of a planning problem, describing a finite set of *objects* of the environment, the *initial state*, and the *goal state* which an agent wishes to achieve.

Definition 4 (Planning Instance). A planning instance is a triple $\Pi = \langle \Xi, \mathcal{I}, G\rangle$, where $\Xi = \langle \Sigma, \mathcal{A}\rangle$ is a planning domain definition; $\mathcal{I} \subseteq \Sigma$ is the initial state specification, defined by specifying the

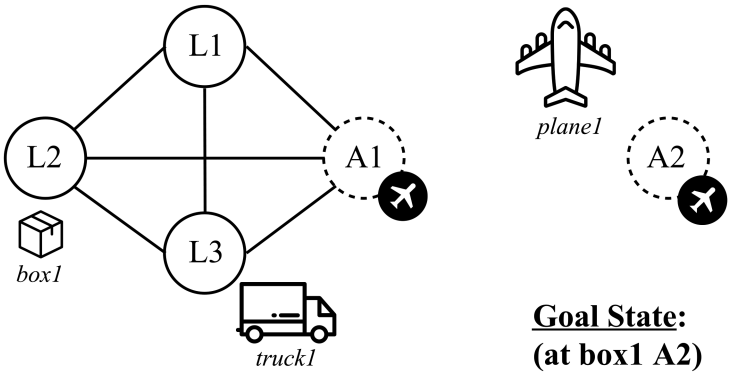


Fig. 1. Logistics problem example.

value for all facts in the initial state; and $G \subseteq \Sigma$ is the goal state specification, which represents a desired subset of facts to be achieved.

Definition 5 (Plan). Let $\Pi = \langle\langle \Sigma, \mathcal{A}\rangle, \mathcal{I}, G\rangle$ be a planning instance. A plan π for Π is a sequence of applicable actions $[a_1, a_2, \ldots, a_n]$ (where $a_i \in \mathcal{A}$) that modifies the initial state $\mathcal{I}$ into one in which the goal state G holds by the successive (ordered) execution of actions in a plan π such that the preconditions of actions $[a_1, a_2, \ldots, a_n]$ are satisfied throughout the execution of the plan π (i.e., $\gamma(\gamma(\gamma(\ldots\gamma(\mathcal{I}, a_1)\ldots), a_{n_1}), a_n) \models G$).

Planners often exploit heuristics which estimate the cost to achieve a specific goal from some state [12]. In this work, and as done by many other classical planners, we consider that all actions have equal cost, making the cost of a plan equal to its length. When a heuristic never overestimates the cost to achieve a goal, it is called *admissible* and guarantees optimal plans when used with certain planning algorithms. In this work, we use both admissible and inadmissible domain-independent heuristics for estimating the distance to a monitored goal.

2.2 Landmarks

Planning landmarks are necessary properties (or actions) that must be true (or executed) at some point in every valid plan (c.f. Definition 5) to achieve a particular goal from an initial state. Hoffman et al. [15] define landmarks as follows.

Definition 6 (Fact Landmarks). Given a planning instance $\Pi = \langle \Xi, \mathcal{I}, G\rangle$, a formula L is a landmark in Π iff L is true at some point along all valid plans that achieve a goal state G from an initial state $\mathcal{I}$.

Hoffmann et al. [15] describe both conjunctive and disjunctive landmarks. A conjunctive landmark is a set of facts that must be true together at some state in every valid plan to achieve a goal. A disjunctive landmark is a set of facts in which one of facts must be true at some state in every valid plan to achieve a goal. Landmarks are often partially ordered by their pre-requisite dependencies. The process of landmark extraction both identifies conjunctive and disjunctive landmarks, and determines the temporal ordering between them (i.e., identifies which landmark occurs before which). As an example of landmarks and their orderings, consider an instance of the Logistics[1] planning problem shown in Figure 1. This example shows two cities: the city on the left contains

[1]Logistics is a domain (adapted from Long et al. [21]) that consists of airplanes and trucks transporting packages between locations (e.g., airports and cities).

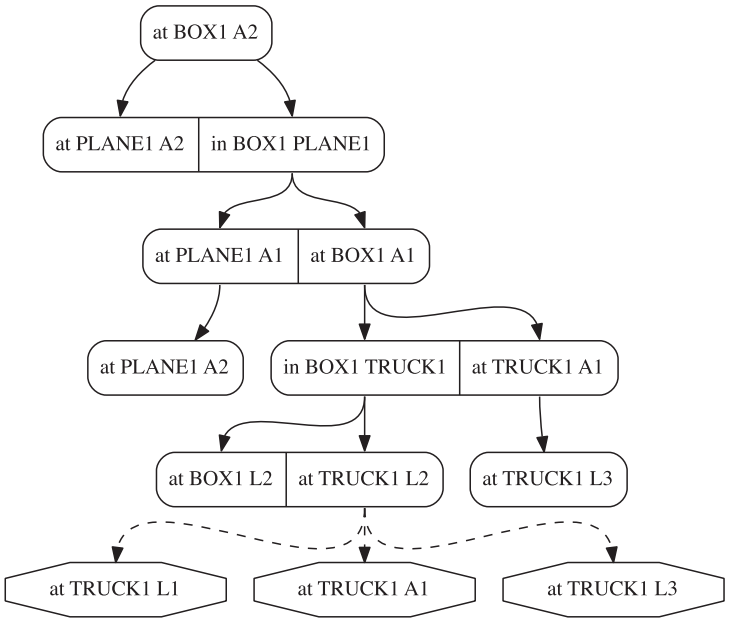


Fig. 2. Ordered fact landmarks extracted from the Logistics example from Figure 1. Fact landmarks that must be true together are represented by connected boxes and represent conjunctive landmarks. Disjunctive landmarks are represented by octagonal boxes connected by dashed lines.

locations L1 through L3 and an airport (A1), and the city on the right contains another airport (A2). The goal within this example is to transport an item (box1) from location L2 to location A2. Listing 1 shows the resulting fact landmarks, whereas Figure 2 shows their ordering (edges show that a source formula must hold after its target). Note that a goal is considered to be a conjunctive landmark. From Figure 2, we see that the second landmark (stating that any valid plan must have box1 within plane1, and that plane1 must be at airport A2) must occur before the goal is achieved, and that before the box is within the plane, plane1 must be at A1, and so must box1.

```
Fact Landmarks:
(and (at BOX1 A2))
(and (at PLANE1 A2) (in BOX1 PLANE1))
(and (at PLANE1 A1) (at BOX1 A1))
(and (at PLANE1 A2))
(and (at TRUCK1 L3))(and (at TRUCK1 L3))
(and (in BOX1 TRUCK1) (at TRUCK1 A1))
(and (at BOX1 L2) (at TRUCK1 L2))
(or (at TRUCK1 L1) (at TRUCK1 A1) (at TRUCK1 L3))
```

Listing 1. Fact landmarks (conjunctive and disjunctive) extracted from the Logistics example.

Whereas in automated planning the concept of landmarks is used to build heuristics [7, 19, 36] and as a fundamental part of planning algorithms [13, 37], in this work we propose to use landmarks to monitor an agent's plan execution and detect sub-optimal actions in a plan. Intuitively, we use landmarks as waypoints (or stepping stones) to monitor what states (or actions) an agent cannot avoid while seeking to achieve its goal.

In the planning literature, there are several algorithms to extract landmarks and their orderings [19, 36, 45], and in this work, for extracting landmarks from planning instances we use the algorithm of Hoffmann et al. [15]. We note that many landmark extraction techniques, including that of Hoffmann et al. [15], have the potential to infer incorrect landmark orderings, which can lead to problems if the optimality monitoring problem relies on the ordering information to make inferences. Nevertheless, our empirical evaluation shows that landmark orderings do not affect detection performance in our experimental dataset, and we discuss landmark orderings later in the article.

2.3 Fact Partitioning

To perform goal recognition, Pattison and Long [26] classify facts into mutually exclusive partitions so as to infer whether certain observations are likely to be goals. Their classification relies on the fact that—in some planning domains—predicates may provide additional information that can be extracted by analysing preconditions and effects in operator definitions. Given a set of candidate goals, we use this classification to infer whether certain observations are consistent with a particular goal. If an inconsistency is detected, we can eliminate the candidate goal. Pattison and Long's classification can be formalised as follows.

Definition 7 (Strictly Activating). A fact f is strictly activating if $f \in \mathcal{I}$ and $\forall a \in \mathcal{A}$, $f \notin \mathit{eff}(a)^+ \cup \mathit{eff}(a)^-$. Furthermore, $\exists a \in \mathcal{A}$ such that $f \in \mathit{pre}(a)$.

Definition 8 (Unstable Activating). A fact f is unstable activating if $f \in \mathcal{I}$ and $\forall a \in \mathcal{A}$, $f \notin \mathit{eff}(a)^+$ and $\exists a, b \in \mathcal{A}$, $f \in \mathit{pre}(a)$ and $f \in \mathit{eff}(b)^-$.

Definition 9 (Strictly Terminal). A fact f is strictly terminal if $\exists a \in \mathcal{A}$ such that $f \in \mathit{eff}(a)^+$ and $\forall a \in \mathcal{A}$, $f \notin \mathit{pre}(a)$ and $f \notin \mathit{eff}(a)^-$.

A *Strictly Activating* fact (Definition 7) appears as a precondition and does not appear as an add or delete effect in an operator definition. Unless defined in the initial state, such a fact can never be added or deleted by an operator. An *Unstable Activating* fact (Definition 8) appears as both a precondition and a delete effect in two operator definitions, so once deleted, this fact cannot be re-achieved. The deletion of an unstable activating fact may prevent a plan execution from achieving a goal. A *Strictly Terminal* fact (Definition 9) does not appear as a precondition of any operator definition and, once added, cannot be deleted. For some planning domains, this kind of fact is most likely to be in the set of goal facts, because once added in the current state, it cannot be deleted and remains true until the final state.

The algorithm we describe in the next section utilises fact partitioning to improve its performance by determining whether a goal is or is not achievable. We note that detecting fact partitions depends on the planning domain definition and, more specifically, operator definition. For example, consider an *Unstable Activating* fact. If an action deletes this fact from the current state, it cannot be re-achieved, and any goals which depend on this fact (i.e., for which it is a landmark) are unreachable. However, the presence or absence of fact partitions is highly domain dependent. For example, from the Blocks-World[2] domain, it is not possible to extract any fact partitions, whereas the Easy-IPC-Grid[3] domain contains *Strictly Activating* and *Unstable Activating* facts. Therefore, although our algorithm can exploit fact partitions, they are not required for the algorithm's operation.

[2]Blocks-World is a classical planning domain where a set of stackable blocks must be re-assembled on a table [12, Chapter 2, page 50] (also appeared in Long et al. [21]).
[3]Easy-IPC-Grid is a domain that consists of an agent that moves in a grid using keys to open locked locations (adapted from Long et al. [21]).

2.4 Commitments

Commitments have been used in multi-agent systems to enable autonomous agents to communicate and coordinate successfully to achieve a particular goal [2, 23, 42]. A commitment `C(DEBTOR, CREDITOR, antecedent, consequent)` formalises that the agent `DEBTOR` commits to agent `CREDITOR` to bring about the `consequent` if the `antecedent` holds. Here, the antecedent and consequent conditions are conjunctions or disjunctions of events and possibly other commitments.

In this work, we aim to monitor the `DEBTOR`'s behaviour (i.e., sequence of actions, a plan) to detect if this agent is individually committed to carrying out a plan to achieve the consequent for the `CREDITOR`. In Section 4.2, we detail how we formalise commitments making an analogy to automated planning, much like the work of Meneguzzi et al. [23], and how we detect commitment abandonment by combining the techniques developed in this article.

3 MONITORING AND DETECTING PLAN OPTIMALITY

We now describe our plan optimality monitoring approach that uses landmarks, fact partitioning, and domain-independent heuristics. Intuitively, this approach aims to detect which actions in the execution of an agent plan do not contribute to the plan (*sub-optimal actions*) for achieving the monitored goal. We begin by formalising the notion of plan optimality. Then, we describe a method that uses heuristics to estimate the distance to some monitored goal for every observed action in the plan execution, and infer whether there is any deviation in the observed plan to achieve the monitored goal. Following this, we develop a method that uses landmarks to anticipate what action the observed agent has to perform in the next observation to reduce the estimated distance to the next landmarks, and consequently to the monitored goal. Finally, we describe how plan optimality monitoring can be performed by bringing together these two previous methods.

3.1 Plan Optimality Monitoring Problem

We define plan optimality monitoring as the process of monitoring the execution of a plan by an agent to solve a planning instance (Definition 4) and detecting when the agent executes steps that deviate from any one of the optimal plans which exist for the planning instance [31]. Formally, we want to detect when the observed agent fails to execute one of the optimal plans of Definition 10, and instead executes any of the (possibly infinite) number of valid sub-optimal plans.

Definition 10 (Optimal Plan). Let $\pi = [a_1, \ldots a_n]$ be a plan with length $|\pi| = n$ for a domain Π. We say π is optimal, also written as π^*, if there exists no other plan $\pi^<$ such that $|\pi^<| < |\pi^*|$.

When an agent executes a plan in an environment it is called *plan execution*, formally defined in Definition 11, and this work, such an execution generates an *observation sequence*, formalised in Definition 12. In fully observable environments, there is a one-to-one correspondence between the actions in a plan and observations.

Definition 11 (Plan Execution). A plan execution π_E is the execution of a sequence of applicable actions (i.e., a plan $\pi = [a_1, \ldots a_n]$) from an initial state $\mathcal{I}$ to a particular state. A plan execution π_E can be either optimal or sub-optimal depending on the plan.

Definition 12 (Observation Sequence). Let O be a sequence $[o_1, o_2, \ldots, o_n]$ of observations of a plan's execution with each observation $o_i \in O$ such that $o_i = name(a)$ for some $a \in \mathcal{A}$ (i.e., the name of some instantiated action in the set of actions in a domain definition Ξ).

Intuitively, given a planning instance, we want to detect exactly which actions and their sequence in the plan execution do not contribute towards the monitored goal in a planning instance. We formally define the task of plan optimality monitoring in Definition 13 and note that for this

task, we consider that we always have full observability, so we observe all actions during a plan execution.

Definition 13 (Plan Optimality Monitoring Problem). A plan optimality monitoring problem is a tuple $T_{\pi^*} = \langle \Pi, O \rangle$, where (i) $\Pi = \langle \Xi, \mathcal{I}, G \rangle$ is a planning instance with domain definition $\Xi = \langle \Sigma, \mathcal{A} \rangle$, an initial state $\mathcal{I}$, and goal G, and (ii) $O = \langle o_1, o_2, \ldots, o_n \rangle$ is an observation sequence of the plan execution.

In solving the plan optimality monitoring problem, we seek those actions in the observation sequence that do not contribute to the achievement of the monitored goal G from the initial state $\mathcal{I}$. To do so, we formalise *contributing actions* in Definition 14. Informally, a *contributing action* is identified by selecting, from the set of optimal plans Π^*, that which maximally matches the sequence of observations, and choosing actions from this plan that lead to the goal state. Thus, the *non-contributing actions* are those that are found in the observations but which diverge from the optimal plan. More formally, the sub-optimal actions in O are the actions in $O - \pi^*$.

Definition 14 (Contributing Action). Let $T_{\pi^*} = \langle \langle \Xi, \mathcal{I}, G \rangle, O \rangle$ be a plan optimality monitoring problem with an associated set of optimal plans Π^*. We define the *sequence* of contributing actions C with regards to a plan π as follows:

$$C(\langle \langle \Xi, \mathcal{I}, G \rangle, O \rangle, \pi) = \begin{cases} \langle o_1 \rangle + C(\langle \langle \Xi, \gamma(\mathcal{I}, o_1), G \rangle, \langle o_2, \ldots, o_n \rangle \rangle) & \text{if } o_1 \in \pi \\ C(\langle \langle \Xi, \gamma(\mathcal{I}, o_1), G \rangle, \langle o_2, \ldots, o_n \rangle \rangle) & \text{if } o_1 \notin \pi \\ \langle\ \rangle & otherwise. \end{cases}$$

Here, + denotes the concatenation of two sequences. We say O is consistent with an optimal plan $\pi^* \in \Pi^*$ if and only if $\pi^* = \arg\max_{\pi \in \Pi^*} |C(\langle \langle \Xi, \mathcal{I}, G \rangle, O \rangle, \pi)|$ and note that there may be multiple such plans for a given domain.

Note that the definition of contributing action does allow us to solve this problem in an online fashion—that is, to evaluate observations as the observer perceives them and computing whether these actions are optimal or not.

Example 1. Consider the Logistics example in Figure 1, which shows a planning problem with two cities: the city on the left that contains four locations, L1 through L3, and the city on the right that contains only location A2. Locations A1 and A2 are airports. The goal of this example is to transport the box located at location L2 to location A2. From this planning problem, Table 1 shows the execution of two possible plans: an optimal and a sub-optimal. In the sub-optimal plan execution, grey actions are those that do not contribute to achieve the goal (i.e., sub-optimal actions). Light-grey actions are those that must be taken to undo the non-contributing actions and thus ultimately achieve the goal.

3.2 Analysing Plan Execution Deviation

We now develop a method that analyses a plan execution to identify plan deviation for achieving a goal state from an initial state. To analyse possible plan execution deviation in an observation sequence (Definition 12), we compute the estimated distance to the monitored goal for every state resulting from the execution of an observed action. Note that we assume full plan observability in the sense that no actions are missing from the observations from the initial state up to a time point. For example, if a optimal plan to goal G has 10 action steps, and we observe just four actions in the observation sequence, then $O_G = [o_1, o_2, o_3, o_4]$ corresponds exactly to the four first actions in the plan.

Given a state s, a heuristic h returns an estimated distance $h(s)$ to the goal state [5]. For example, consider the states s_i and s_{i-1} and the observed action o_i at the step i. If the observed action o_i at

Table 1. Plan Optimality Monitoring Example

Optimal Plan	**Sub-Optimal Plan**
0 - (drive TRUCK1 L3 L2 CITY1)	0 - (drive TRUCK1 L3 L2 CITY1)
1 - (loadTruck BOX1 TRUCK1 L2)	1 - (loadTruck BOX1 TRUCK1 L2)
2 - (drive TRUCK1 L2 A1 CITY1)	2 - (unloadTruck BOX1 TRUCK1 L2)
3 - (unloadTruck BOX1 TRUCK1 A1)	3 - (drive TRUCK1 L2 L1 CITY1)
4 - (fly PLANE1 A2 A1)	4 - (drive TRUCK1 L1 L2 CITY1)
5 - (loadAirPlane BOX1 PLANE1 A1)	5 - (loadTruck BOX1 TRUCK1 L2)
6 - (fly PLANE1 A1 A2)	6 - (drive TRUCK1 L2 A1 CITY1)
7 - (unloadAirplane BOX1 PLANE1 A2)	7 - (unloadTruck BOX1 TRUCK1 A1)
	8 - (fly PLANE1 A2 A1)
	9 - (loadAirPlane BOX1 PLANE1 A1)
	10 - (fly PLANE A1 A2)
	11 - (unloadAirplane BOX1 PLANE1 A2)

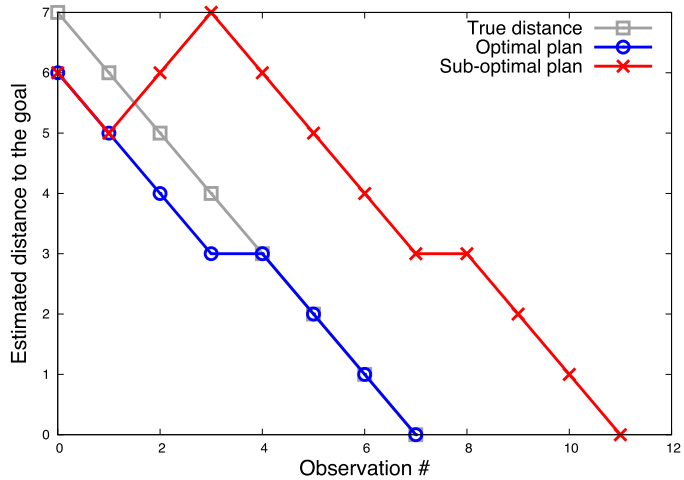


Fig. 3. Plan execution deviation example using the Fast-Forward heuristic.

step i transitions the system to state s_i, we consider a deviation from a plan to occur if $h(s_{i-1}) < h(s_i)$. Such deviations can arise for a variety of reasons including concurrent or interleaved plan execution by the agent (e.g., in an attempt to achieve multiple goals simultaneously), non-optimal plan selection (e.g., due to bounded rationality), and incorrect estimates by the heuristic. The uptick shown in Figure 3 illustrates a deviation detected using the Fast-Forward heuristic [14] for two different plan executions. These two plan executions (an optimal plan (points denoted as circles) and a sub-optimal plan (points denoted as crosses)) are plans that achieve the goal state from the initial state in Figure 1. During the execution of the sub-optimal plan, deviations occur for actions leading at the observation time 2 and 3. By analysing this plan deviation, we conclude that these actions do not contribute to achieve the goal because they increase the estimated distance to the goal state. However, given the potential large deviations from sub-optimal plans combined with the varying ways in which heuristics can inaccurately measure the distance towards a goal, we cannot rely exclusively on deviations from the heuristic to detect sub-optimal actions.

Thus, since heuristics may be inaccurate, we use landmarks to build a further condition of sub-optimality, predicting actions that achieve next landmarks, and consequently the monitored goal state.

3.3 Predicting Upcoming Actions via Landmarks

Ordered landmarks [15] effectively provide waypoints towards a monitored goal from an initial state, and based on these waypoints we can infer what cannot be avoided on the way to achieving such monitored goal. Note that the initial and goal state are themselves landmarks, as all plans begin and terminate in these states. Since all plans should pass through a landmark, we can exploit their presence to predict what actions might be executed next, either to reach the ordered landmarks or to progress towards a monitored goal. We use such predictions to check the set of observed actions of a plan execution to determine which actions do not contribute to achieve the monitored goal [31]. The use of lookahead actions is quite common in AI Planning and search [20, 44]; however, in this article, we use such lookahead (or predictions) to verify if the plan execution is progressing towards the monitored goal without deviating. We formalise this in Algorithm 1.

To predict which actions could reasonably be executed in the next observation and to minimise the accumulated discrepancies due to the imprecise nature of the heuristics, our algorithm identifies the closest landmarks by estimating the distance to the landmarks from the current state. Our approach uses an admissible domain-independent heuristic to estimate the distance to landmarks, namely the Max-Heuristic (or Max-Cost heuristic), which we denote as h_{max}. This heuristic, originally defined by Bonet and Geffner [5] consists of computing the costs of achieving each individual literal g_i in a goal G as follows[4]:

$$h_{max}(s, G) = \max_{g_i \in G} h_{max}(s, g_i)$$

$$h_{max}(s, g_i) = \begin{cases} 0, & \text{if } g_i \in s, \\ \min\{h_{max}(s, a) | a \in \mathcal{A} \text{ and } g_i \in \mathit{eff}(a)\}, & \text{otherwise;} \end{cases}$$

$$h_{max}(s, a) = cost(a) + h_{max}(s, pre(a)).$$

We consider that the neighbouring fact landmarks are those that return estimated distance $h_{max}(s, l) = 0$ and $h_{max}(s, l) = 1$. We use these neighbouring landmarks to identify—where possible—a set of actions which should be executed next. The resulting algorithm (Algorithm 1) iterates over a set of ordered fact landmarks $\mathcal{L}$ (line 3), and, for each landmark l, the Max-Heuristic estimates the distance from the current state s to l. If the estimated distance to landmark l is $h_{max}(s, l) = 0$ (line 5), this means that landmark l is in the current state, and the algorithm selects those actions that contain l as a precondition, because these can be executed immediately (line 6). Otherwise, if the estimated distance to landmark l is $h_{max}(s, l) = 1$ (line 7), this means that landmark l can be reached by executing a single action, and the algorithm selects those actions that are applicable in the current state and contain l as an effect (line 8). These actions are selected because they reduce the distance to the next landmark, and consequently to the monitored goal. Thus, we use the observed plan execution to estimate which actions do not contribute to achieve a goal. Example 2 shows how Algorithm 1 predicts upcoming actions using landmarks.

Example 2. Consider the Logistics problem in Figure 1. If the current state is the initial state, then the algorithm predicts upcoming actions that might be executed as the first observation in the plan execution. Table 2 shows three columns: (1) the set of fact landmarks to achieve the goal from the initial state (Landmarks), (2) the estimated distance from the current state (i.e., initial

[4]Since we assume unit-cost actions, $cost(a) = 1$.

Table 2. Predicted Upcoming Actions for the Logistics Example in Figure 1

Landmarks	$h_{max}(s, l)$	**Predicted Actions**
`(and (at BOX1 A2))`	7	-
`(and (at PLANE1 A2) (in BOX1 PLANE1))`	6	-
`(and (at PLANE1 A1) (at BOX1 A2))`	5	-
`(and (at PLANE1 A2))`	0	`(fly PLANE1 A2 A1)`
`(and (at TRUCK1 L3))`	0	`(drive TRUCK1 L3 L2 CITY1)`
`(and (in BOX1 TRUCK1) (at TRUCK AIRPORT-C))`	3	-
`(and (at BOX1 L2) (at TRUCK1 L2))`	1	-
`(or`		
`(at TRUCK1 L1)`	1	-
`(at TRUCK1 A1)`	1	-
`(at TRUCK1 L3))`	0	`(drive TRUCK L3 L2 CITY1)`

state) to fact landmarks using $h_{max}(l)$, and (3) which applicable actions our algorithm predicts to be in the next observation (Upcoming Actions). From these landmarks, three of them have $h_{max}(s, l) = 0$, namely `(at PLANE1 A1)`, `(at TRUCK1 L3)`, and `(at TRUCK1 L3)`, and other three have $h_{max}(s, l) = 1$, namely `(and (in BOX1 TRUCK1) (at TRUCK1 A1))`, `(at TRUCK1 L1)`, and `(at TRUCK1 A1)`. Note that there are no upcoming (predicted) actions for the fact landmarks for which the estimated distance is $h_{max}(s, l) = 1$, because there is no applicable action in the initial state to achieve these fact landmarks. Thus, from the landmarks which have the estimated distance $h_{max}(s, l) = 0$, Algorithm 1 predicts two actions as the first expected observation: `(fly PLANE1 A2 A1)` or `(drive TRUCK1 L3 L2 CITY1)`. These actions aim to reduce the distance to the next ordered landmarks, and consequently to the monitored goal.

3.4 Detecting Sub-Optimal Action Steps

We now develop our approach to detect sub-optimal action steps [31], bringing together the methods that were presented in Sections 3.2 and 3.3. Algorithm 2 formally describes our planning-based

ALGORITHM 1: Compute Upcoming Actions via Landmarks.

Parameters: $\Xi = \langle \Sigma, \mathcal{A} \rangle$ *planning domain*, s *current state*, and $\mathcal{L}$ *ordered fact landmarks.*
Output: $\eta_{PActions}$ *set of possible upcoming actions.*

1: **function** PredictUpcomingActions(Ξ, s, $\mathcal{L}$)
2: $\eta_{PActions} \leftarrow \emptyset$
3: **for** each fact landmark l in $\mathcal{L}$ **do**
4: $Al \leftarrow \emptyset$
5: **if** $h_{max}(s, l) = 0$ **then** ▷ $h_{max}(s, l)$ *estimates* l *from* s.
6: $Al \leftarrow$ all a in $\mathcal{A}$ s.t. $l \in pre(a)$
7: **else if** $h_{max}(s, l) = 1$ **then**
8: $Al \leftarrow$ all $a \in \mathcal{A}$ s.t $pre(a) \in s \wedge l \in eff(a)^+$
9: **end if**
10: $\eta_{PActions} := \eta_{PActions} \cup Al$
11: **end for**
12: **return** $\eta_{PActions}$
13: **end function**

approach to detect sub-optimal plan steps. The algorithm takes as input a plan optimality monitoring problem T_{π^*} (Definition 13)—that is, a planning domain, an initial state, a monitored goal, and a set of observed actions as the execution of an agent plan. The algorithm initially computes key information using the landmark extraction algorithm proposed by Hoffman et al. [15] (using the function ExtractLandmarks). Afterwards, it analyses plan execution by iterating over the set of observed actions and applying them, checking which actions do not contribute to the monitored goal. Any such action that does not contribute to achieve the monitored goal is then considered to be sub-optimal. When analysing plan execution deviation (via the distance to the monitored goal), our algorithm can use any domain-independent heuristic, and to do so, we estimate goal distance using the function EstimateGoalDistance.[5] We use Algorithm 1 (PredictUpcomingActions) to predict upcoming actions via landmark consideration, in turn utilising Max-Heuristic due to its admissibility, and the fact that it estimates costs for only a short distance (0 or 1). The "if" statement on line 10 combines heuristic cost estimation and landmark based action prediction, labelling a step as sub-optimal if an observed action is not in the set of predicted upcoming actions and the estimated distance of the current state is greater than the previous one.

ALGORITHM 2: Plan Optimality Monitoring.

Parameters: $\Xi = \langle \Sigma, \mathcal{A} \rangle$ *planning domain*, $\mathcal{I}$ *initial state*, G *monitored goal*, and O *observed actions*.
Output: $A_{sub-optimal}$ *as sub-optimal actions*.

```
1:  function MonitorPlanOptimality(Ξ, I, G, O)
2:      A_sub-optimal ← ∅            ▷ Actions that do not contribute to achieve the monitored goal G.
3:      L ← ExtractLandmarks(I, G)
4:      s ← I                                                    ▷ s is the current state.
5:      η_PActions ← PredictUpcomingActions(Ξ, s, L)
6:      D_G ← EstimateGoalDistance(s, G)   ▷ A domain-independent heuristic to estimate goal
        G from s.
7:      for each observed action o in O do
8:          s ← s.Apply(o)
9:          D'_G ← EstimateGoalDistance(s, G)
10:         if o ∉ η_PActions ∧ (D'_G > D_G) then
11:             A_sub-optimal ← A_sub-optimal ∪ o
12:         end if
13:         η_PActions ← PredictUpcomingActions(Ξ, s, L)
14:         D_G ← D'_G
15:     end for
16:     return A_sub-optimal
17: end function
```

We note that our approach iterates over the observation sequence O after extracting landmarks for G, and during each iteration, and also iterates over all fact landmarks $\mathcal{L}$ to predict non-regressive actions, and after, it calls a heuristic function to estimate the distance to G. If the complexity of extracting landmarks is EL, and of running the heuristic function HF, then the complexity of our plan optimality monitoring approach is bounded by $O(EL + |O| \cdot |\mathcal{L}| \cdot HF)$.

[5] In Section 5.3, we provide a list of domain-independent heuristics that we used in our experiments to estimate goal distance.

4 DETECTING COMMITMENT ABANDONMENT

In this section, we apply our approach for plan optimality monitoring to infer when agents are likely to abandon commitments to each other. Consider the following scenario. An agent A_1 delegates a goal G_1 to an agent A_2 such that A_2 is now committed to achieving G_1, and consider that G_1 can be achieved optimally through a plan consisting of actions $o_1^{G1}, o_2^{G1}, o_3^{G1}$. However, instead of executing any of these actions, A_2 executes $o_1^?, o_2^?, o_1^{G1}, o_3^?$. In this situation, agent A_1 needs to determine whether or not A_2 is still committed to G_1 or not, and decide whether to assume A_2 abandoned G_1, and delegate G_1 to another agent. Motivated by such scenario, we formally define the commitment abandonment problem and then develop an approach to efficiently solve this problem using fact partitions (Section 2.3) and the techniques from Section 3.

4.1 Commitment Abandonment Problem

We define *commitment abandonment* as a situation in which an agent switches from executing the actions of one plan that achieves the consequent it is committed to by executing actions from another plan. This plan may achieve other goals, including the consequent of other commitments, or the agent has no intention to achieve its original commitment. Actions in a plan that do not contribute to achieve the consequent of a commitment may indicate that the debtor agent is likely to abandon this commitment. An agent may abandon a commitment for a variety of reasons. For example, it may have conflicting commitments and must abandon the less important commitment to achieve the more important one.

Here, we take inspiration from earlier work [2, 23] that connects commitments to planning, so the domain definition Ξ represents the environment where agents can interact and act (i.e., Σ is set of environment properties and $\mathcal{A}$ is a set of available actions). Now, consider a commitment `C(DEBTOR, CREDITOR, At, Ct)`: for a `DEBTOR` to achieve the consequent `Ct` from the antecedent `At`, we require that (i) the antecedent `At` must be in the initial state $\mathcal{I}$ (i.e., `At` $\subseteq \mathcal{I}$) and (ii) the consequent `Ct` is the goal G. Thus, a plan π for `C(DEBTOR, CREDITOR, At, Ct)` is a sequence of actions $[a_1, a_2, \ldots, a_n]$ (where $a_i \in \mathcal{A}$) that modifies the state `At` $\subseteq \mathcal{I}$ into one where `Ct` holds by the successive (ordered) execution of actions in a plan π. We note that both antecedent `At` and consequent `Ct` consist of a set of facts—more specifically, both are states.

To decide if a debtor will abandon a commitment, we monitor its behaviour in an environment by observing its successive execution of actions. This successive execution of actions represents an observation sequence (Definition 12) that should achieve a consequent from an antecedent. When a `DEBTOR` commits to an agent `CREDITOR` to bring about the `consequent` of a commitment, the `DEBTOR` should individually commit to achieving such a `consequent` state, and to achieve such state, the `DEBTOR` has to execute a plan. An observer does not have access to `DEBTOR`'s internal state, and consequently to what plan it has committed to. Therefore, when there are multiple optimal plans, we need to be able to determine which of those plans the `DEBTOR` is pursuing. Thus, in Definition 15, we formally define an individual commitment from an observer's point of view.

Definition 15 (Individual Commitment). Given a set of plans, a `DEBTOR` agent is individually committed to a plan π if, given a sequence of observations $o_1, \ldots, o_m$, (i) $o_k \in \pi$ where $(1 \leq k \leq m)$ and (ii) if $o_k = a_j$, then $\forall i = 1 \ldots j-1$, $a_i \in O$ and a_i occurs before a_{i+1} in O. An observation o_p does not contribute to achieve a consequent `Ct` if the `DEBTOR` agent is committed to plan π and $o_p \notin \pi$, or if $o_p = a_j$, a_k has not yet been observed where $k < j$.

Finally, using the notion of an individual commitment, we formally define a commitment abandonment problem over a planning theory in terms of a large enough deviation from such an individual commitment in Definition 16. Note that with Definition 15, we can now think of deviations

from observations that constitute a strict sub-sequence of any optimal plan that start with the initial state, allowing an agent to infer abandonment at any point in a partial plan execution.

Definition 16 (Commitment Abandonment Problem). A commitment abandonment problem is encoded as a tuple $CA = \langle \Xi, C, \mathcal{I}, O, \theta \rangle$, in which Ξ is a planning domain definition; C is the commitment, in which C(DEBTOR, CREDITOR, At, Ct), DEBTOR is the debtor, CREDITOR is the creditor, At is the antecedent condition, and Ct is the consequent. $\mathcal{I}$ is the initial state (s.t., At $\subseteq \mathcal{I}$), O is an observation sequence of the plan execution with each observation $o_i \in O$ being an action from domain definition Ξ, and θ is a threshold that represents the permitted fraction of actions (in relation to an individually committed from Definition 15) in an observation sequence that do not contribute to achieving Ct from At $\subseteq \mathcal{I}$ in which the DEBTOR can execute in O.

The solution for a commitment abandonment problem is whether an observation sequence O has deviated more than θ from the optimal plan to achieve the consequent Ct of commitment C. In this work, we use the threshold θ as a degree of tolerance to sub-optimality that the creditor affords the debtor.

4.2 Detecting Commitment Abandonment via Optimality Monitoring

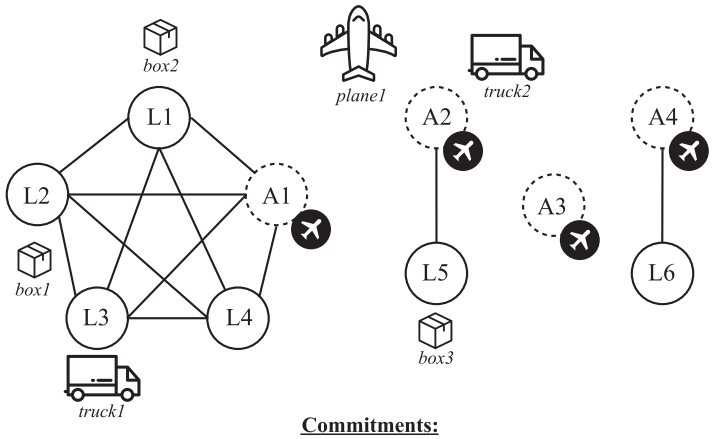
To detect commitment abandonment, we infer sub-optimal action steps combining the techniques from Section 3 and use the concept of fact partitions from Section 2.3. Once we observe evidence of such fact partitions in the observations, we can determine that a goal is no longer achievable. We extract fact partitions using a function called PartitionFacts.

Algorithm 3 formalises our approach to solve a commitment abandonment problem. The algorithm takes as input a commitment abandonment problem *CA* and returns whether a commitment has been abandoned, based on whether one of the following occurs during plan execution: (1) if *Strictly Activating* facts that we extracted are not in the initial state (line 3), (2) if we observe the evidence of any *Unstable Activating* and *Strictly Terminal* facts during the execution of actions in the observations (line 8), or (3) if the number of sub-optimal action steps are greater than the threshold θ (i.e., the percentage of actions away from optimal execution that the creditor allows the debtor to deviate in achieving the consequent state) defined by the creditor (line 12). If none of these conditions hold, the debtor is considered to remain committed to achieving the consequent state of the commitment. Note that in condition (2), the presence of predicates from two of the fact partitions can determine that the monitored goal (or consequent) is unreachable, because there is no available action that can make the facts hold.

The checks from condition (3) are substantially different from the other conditions, since this is a *subjective* measure of abandonment, which we capture in the threshold θ. We need to use this subjective measure because in most realistic domains, there is never a definite logical condition (which is what we capture with the fact partitions) that tells that the commitment is irreparably abandoned, and this threshold allows us to reason about such non-clear-cut situations. One can think about the threshold as a subjective measure of *patience* on the part of the creditor of a commitment. Specifically, this captures the amount of slack given to a debtor when the debtor interleaves its other goals with the delegated one.

4.3 Working Example

To exemplify how our approaches detect sub-optimal action steps and determine commitment abandonment, consider the Logistics problem example shown in Figure 4. This example formalises two commitments: C1 represents that the debtor agent TRUCK1 is committed to the creditor agent PLANE1 to bring about the consequent (at BOX3 L1) when the antecedent (at BOX3 A1) becomes true, and C2 represents that the debtor agent PLANE1 is committed to the creditor agent



Commitments:

C1: C(TRUCK1, PLANE1, (at BOX3 A1), (at BOX3 L1))

C2: C(PLANE1, TRUCK1, (and (at BOX1 A1) (at BOX2 A1)), (and (at BOX1 A3) (at BOX2 A4)))

Fig. 4. Logistics working example.

TRUCK1 to bring about the consequent `(and (at BOX1 A3) (at BOX2 A4))` when the antecedent `(and (at BOX1 A1) (at BOX2 A1))` becomes true. Assuming that for C1 the threshold θ is 0, and for C2 the threshold θ is 0.3, Tables 3 and 4 show observed actions for C1 and C2, respectively. Rows in grey represent sub-optimal actions, and rows without a number (i.e., -) represent actions executed by the creditor agent that are going to achieve the antecedent state.

ALGORITHM 3: Detecting Commitment Abandonment via Plan Optimality Monitoring and Fact Partitions.

Parameters: $\Xi = \langle \Sigma, \mathcal{A} \rangle$ *planning domain*, At *antecedent condition* ($At \subseteq \mathcal{I}$), Ct *consequent condition*, $\mathcal{I}$ *initial state*, O *observation sequence*, and θ *threshold.*
Output: True or False.

```
1:  function HasAbandoned(Ξ, At, Ct, I, O, θ)
2:      ⟨F_sa, F_ua, F_st⟩ ← PartitionFacts(Σ, A)
3:      if F_sa ∩ (At ⊆ I) = ∅ then
4:          return true                                  ▷ Ct is no longer possible.
5:      end if
6:      for each observed action o in O do
7:          s ← s.Apply(o)
8:          if (F_ua ∪ F_st) ⊆ s then
9:              return true                              ▷ Ct is no longer possible.
10:         end if
11:     end for
12:     A_sub−optimal ← MonitorPlanOptimality(Ξ, I, Ct, O)
13:     if |A_sub−optimal| > (θ * |O|) then
14:         return true                                  ▷ Debtor has abandoned the commitment.
15:     end if
16:     return false                                     ▷ Debtor may still be committed to achieve Ct.
17: end function
```

Table 3. Observation Sequence (1), Monitoring TRUCK1

-	(loadAirplane BOX3 PLANE1 A2)
-	(fly PLANE1 A2 A1)
-	(unloadAirplane BOX3 PLANE1 A1)
0	(loadTruck BOX3 TRUCK1 A1)
1	(drive TRUCK1 A1 L4 CITY1)
2	(drive TRUCK1 L4 L2 CITY1)
3	(drive TRUCK1 L2 L1 CITY1)

Table 4. Observation Sequence (2), Monitoring PLANE1

-	(drive TRUCK1 L2 A1 CITY1)
-	(unloadTruck BOX2 TRUCK1 A1)
-	(unloadTruck BOX1 TRUCK1 A1)
0	(fly PLANE1 A2 A1)
1	(loadAirplane BOX2 PLANE1 A1)
2	(loadAirplane BOX1 PLANE1 A1)
3	(fly PLANE1 A1 A2)
4	(fly PLANE1 A2 A1)
5	(fly PLANE1 A1 A3)
6	(unloadAirplane BOX1 PLANE1 A3)
7	(fly PLANE1 A3 A4)
8	(unloadAirplane BOX2 PLANE1 A4)

From the observation sequence shown in Table 3 and the threshold $\theta = 0\%$, our approach returns that the observations at times 1 and 2 are sub-optimal actions, and therefore debtor agent TRUCK1 has abandoned commitment C1, since $\theta = 0$ (i.e., the creditor does not allow any deviation), and the agent has executed two actions that do not contribute to achieve the consequent state of C1. The observed action at time 3 is an optimal action because the agent is moving towards the location L1, where it must unload BOX3.

Now consider the observation sequence in Table 4 and a threshold $\theta = 0.3$. Although our approach returns that the observations at times 3 and 4 are sub-optimal actions, the threshold (which allows $9 \times 0.3 = 2.7$ sub-optimal actions) means that the debtor agent PLANE1 is considered to remain committed to achieve the consequent of C2.

5 EXPERIMENTS AND EVALUATION

In this section, we describe the experiments and evaluation we carried out on our approaches. In Section 5.1, we describe the planning domains and the datasets we used as a benchmark to our proposed approaches. In Section 5.2, we show the set of metrics we used for evaluation. In Section 5.3, we describe the domain-independent heuristics we used in our experiments. Finally, in Sections 5.4 and 5.5, we show the experiments and evaluation on our plan optimality monitoring approach and our commitment abandonment approach. To evaluate our approaches, we ran all experiments using a single core of a 12 core Intel Xeon CPU E5-2620 v3 @ 2.40 GHz with 16 GB of RAM. The Java virtual machine we ran the experiments on was limited to 1 GB of memory, and we imposed a 1-minute time-out for all of our experiments.

5.1 Domains and Datasets

We empirically evaluated our approaches (plan optimality monitoring and commitment abandonment detection) over several widely used planning domains, most of which are inspired by real-world scenarios. Most domains we used are from the Artificial Intelligence Planning and Scheduling (AIPS) competitions in 1998, 2000, and 2002 [1, 8, 21], and goal and plan recognition datasets [28]. The Driver-Log domain consists of drivers that can walk between locations and trucks that can drive between locations, and goals consist of transporting packages between locations. Depots combines transportation and stacking, in which goals involve moving and stacking packages by using trucks and hoists between depots. Easy-IPC-Grid consists of an agent that moves in a grid from cells to others by transporting keys to open locked locations for releasing agents that are at isolated cells. The Ferry domain consists of set of cars that must be moved to desired locations using a ferry that can carry only one car at a time. Logistics, described previously, consists of airplanes and trucks transporting packages between locations (e.g., airports and cities). Satellite involves using one or more satellites to make observations by collecting data and downloading the data to a desired ground station. Sokoban involves pushing a set of boxes into specified locations in a grid with walls. Finally, Zeno-Travel is a domain where passengers can embark and disembark onto aircraft that can fly at two alternative speeds between locations. We select and use these domains in our datasets because they are inspired by real-world scenarios, and most of them contain and deal with more than one agent and several objects in the environment.

For each of these domains, we selected 15 to 30 non-trivial problem instances,[6] with each problem instance also associated to a set of observations (i.e., plan executions)—for instance, a plan optimality monitoring problem. This set of observations can represent either an optimal or a sub-optimal plan execution. We generate plans (optimal and sub-optimal) using open-source planners, such as BlackBox, Fast-Downward, FF, and LAMA [37]. For sub-optimal plans, we (manually) annotated the sub-optimal action steps and how many sub-optimal steps each plan has. These steps consist of actions that do not contribute to achieving the monitored goal, representing steps that our plan optimality monitoring approach aims to detect. We manually annotated the sub-optimal steps in the sub-optimal plans to define the exact timesteps in which the sub-optimal actions happened during a plan execution. The key goals in optimality monitoring is to detect not only the *number* of sub-optimal steps (a relatively easy problem) in a plan execution but also the *exact time steps* in which such sub-optimal actions happened in a plan.

We built a dataset for the experiments on detecting commitment abandonment using 30 commitment abandonment problems (10 problems for each threshold value: 0%, 10%, and 30%), and for these problems we generated plans (observed actions) that either abandoned (ultimately went to a different goal or consequent) or did not abandon their corresponding goals/consequents, varying the number of abandoned actions. For instance, there are commitment abandonment problems that contain plans with sub-optimal steps that do not abandon the defined goals/consequents, and it happens because the number of sub-optimal steps is not greater than the permitted fractions of actions (θ) that are allowed to deviate during the plan execution. Like the dataset for plan optimality monitoring, we use non-trivial problem instances to define the commitments in our dataset for commitment abandonment detection, varying the number of observations (i.e., plan length) between 10.0 and 23.5 ($|O|$). Our dataset ensures that the extra actions added to all sub-optimal plans do not make the actual monitored goal unreachable or invalidate the plan.

[6] A non-trivial planning problem contains a large search space (in terms of search branching factor and depth); therefore, even modern planners such as Fast-Downward (FD) take up to 5 minutes to complete. In our datasets, the number of instantiated (grounded) actions is between 146 and 4,322, and plan length is between 12.2 and 25.7.

5.2 Evaluation Metrics

Using the generated datasets, we evaluated how accurately our approaches detect actions that do not contribute to achieve a corresponding monitored goal and commitment abandonment by using the following metrics. *Precision* (PPV (positive predictive value)) is the ratio between true positive results, and the sum of true positive and false positive results. *Precision* provides the percentage of positive predictions that is correct. *Recall* (TPR (true positive rate) is the ratio between true positive results, and the sum of the number of true positives and false negatives. *Recall* provides the percentage of positive cases that our approaches have detected. The *F1-score* (F1) is a measure of accuracy that aims to provide a trade-off between *Precision* and *Recall*.

5.3 Planning Heuristics

Since our approaches can exploit any domain-independent heuristic to compute whether an action contributes to goal achievement, we evaluated our approaches using several admissible and inadmissible heuristics from the planning literature, as follows:

- Max-Heuristic (h_{max}) is an admissible heuristic proposed by Bonet and Geffner [5], and this heuristic is based on the delete-list relaxation, in which delete effects of actions are ignored during calculation of the heuristic cost to a goal. This calculation is the cost of a conjunctive goal, which represents the maximum cost to achieve each of the individual facts.
- Sum (h_{sum}) is also an admissible heuristic proposed by Bonet and Geffner [5], and this heuristic works similarly to Max-Heuristic. However, the Sum heuristic is often more informative than Max-Heuristic.
- Adjusted-Sum (h_{adjsum}) [25] is an inadmissible heuristic that improves the Sum heuristic by taking into account both negative and positive interactions among facts.
- Adjusted-Sum2 ($h_{adjsum2}$) [25] is an inadmissible heuristic that improves its previous version (Adjusted-Sum) by combining the computation of the Set-Level heuristic[7] and the relaxed plan heuristic.
- Adjusted-Sum2M ($h_{adjsum2M}$) [25] is an inadmissible heuristic that improves the Adjusted-Sum2.
- Combo (h_{combo}) [25] is an inadmissible heuristic that improves the Adjusted-Sum by combining the computation of the Adjusted-Sum heuristic and the Set-Level heuristic.
- Fast-Forward (h_{ff}) is a well-known inadmissible heuristic in the planning community [14] that relies on state-space search and estimates the goal distance by using the delete-list relaxation.

5.4 Plan Optimality Monitoring Experiments

For experiments and evaluation on our plan optimality monitoring approach, we use the metrics presented before, as follows: *Precision* (PPV), *Recall* (TPR), and *F1-score* (F1). Here, for detecting sub-optimal action steps, true positive results represent the number of sub-optimal actions detected that do not contribute to achieve the monitored goal. False positive results represent the number of actions that our approach labelled as a sub-optimal action which is in fact an optimal action. False negative is a sub-optimal action that is not detected by our approach.

We separated our experiments in three different parts, evaluating our plan optimality monitoring techniques separately (techniques from Sections 3.2 and 3.3) and together (approach from

[7]The Set-Level heuristic estimates the cost to a goal by returning the level of the planning graph where all facts of the goal sate are reached without any mutexes [24].

Table 5. Experimental Results for Detecting Sub-Optimal Action Steps Using the Technique Predicting Upcoming Actions via Landmarks (1)

Plan Optimality Monitoring (Predicting Upcoming Actions via Landmarks)				
Domain	$\|O\|$	$\|\mathcal{L}\|$	**Time**	**PPV / TPR / F1**
Driver-Log (20)	20.1	53.6	0.19	18.8% / 72.5% / 29.8%
Depots (30)	16.7	64.7	0.33	30.7% / 65.4% / 41.8%
Easy-IPC-Grid (30)	14.1	48.5	0.26	34.9% / 74% / 47.4%
Ferry (30)	13.8	18.1	0.06	41.4% / 96.6% / 58%
Logistics (30)	20.8	24.0	0.20	26.3% / 86.9% / 40.4%
Satellite (20)	25.7	60.8	1.10	13.1% / 70.2% / 22.0%
Sokoban (20)	24.0	76.5	0.49	1.4% / 21.4% / 2.7%
Zeno-Travel (15)	12.2	38.7	0.21	32.7% / 76.9% / 45.9%

Table 6. Experimental Results for Detecting Sub-Optimal Action Steps Using the Technique Analysing Plan Execution Deviation (1)

Plan Optimality Monitoring (Analysing Plan Execution Deviation)								
			h_{adjsum}		$h_{adjsum2}$		$h_{adjsum2M}$	
Domain	$\|O\|$	$\|\mathcal{L}\|$	**Time**	**PPV / TPR / F1**	**Time**	**PPV / TPR / F1**	**Time**	**PPV / TPR / F1**
Driver-Log (20)	20.1	53.6	0.15	63.6% / 77.8% / 70%	0.23	47.2% / 94.4% / 63%	0.65	47.4% / 100% / 64.3%
Depots (30)	16.7	64.7	0.27	46% / 88.5% /60.5%	0.39	47.2% / 96.2% / 63.3%	1.01	47.4% / 96.4% / 63.5%
Easy-IPC-Grid (30)	14.1	48.5	0.28	83.9% / 92.9% / 88.1%	0.37	100% / 96.2% / 98%	0.46	86.4% / 79.2% / 82.6%
Ferry (30)	13.8	18.1	0.13	57.6% / 67.9% / 62.3%	0.17	100% / 78.6% / 88%	0.34	75.0% / 42.9% / 54.5%
Logistics (30)	20.8	24.0	0.13	100% / 13% / 23.1%	0.17	100% / 91.3% / 95.5%	0.59	100% / 91.3% / 95.5%
Satellite (20)	25.7	60.8	0.45	100% / 26.7% / 42.1%	0.53	45.5% / 66.7% / 54.1%	3.21	44% / 73.3% / 55%
Sokoban (20)	24	76.5	1.18	35.5% / 78.6% / 48.9%	1.41	5.4% / 14.3% / 7.8%	3.50	35.5% / 78.6% / 48.9%
Zeno-Travel (15)	12.2	38.7	0.14	77.8% / 50.0% / 60.9%	0.17	82.4% / 100% / 90.3%	0.77	72.2% / 92.9% / 81.3%

Section 3.4), as follows: (1) Table 5 shows experimental results by using just our technique for predicting upcoming actions via landmarks for plan optimality monitoring; (2) Tables 6 and 7 show experimental results by using just our technique for analysing plan execution deviation using all domain-independent heuristics we presented before; and, finally, (3) Tables 8 and 9 show experimental results of our plan optimality monitoring approach combining the prediction of upcoming actions and the plan deviation detection approach (as we developed in Section 3.4). Each row contains results for all domains averaged over all problem instances for the set of observations $|O|$ (i.e., the number of actions in a plan execution—plan length), the average number of extracted fact landmarks $\mathcal{L}$, monitoring time (in seconds), and metrics (*Precision* (PPV), *Recall* (TPR), and the *F1-score* (F1)). The observation averages of column $|O|$ range between 12.2 and 25.7, indicating that all plans we analyse are non-trivial plan executions.

Table 5 shows the results of our optimality monitoring technique for predicting upcoming landmarks. These results show that our technique is quite fast (at most 1 second) and relatively accurate for all domains but Sokoban. Apart from Sokoban, the TPR results are higher than 65.4%. For Logistics and Ferry instances, this technique yields almost perfect results with respect to TPR, but the results for PPV and F1 are quite low, at less than 50% on average. Other domains result in lower FPR and F1 in detecting sub-optimal action steps in a plan. Thus, the technique can be useful on its own in correctly detecting upcoming actions via landmarks, showing good results for

Table 7. Experimental Results for Detecting Sub-Optimal Action Steps Using the Technique Analysing Plan Execution Deviation (2)

Plan Optimality Monitoring (Analysing Plan Execution Deviation)									
			h_{combo}		h_{ff}		h_{sum}		
Domain	$\|O\|$	$\|\mathcal{L}\|$	**Time**	**PPV / TPR / F1**	**Time**	**PPV / TPR / F1**	**Time**	**PPV / TPR / F1**	
Driver-Log (20)	20.1	53.6	0.69	63.6% / 77.8% / 70.0%	0.14	47.2% / 94.4% / 63%	0.16	63.6% / 77.8% / 70%	
Depots (30)	16.7	64.7	1.03	43.6% / 92.3% / 59.3%	0.27	47.2% / 96.2% / 63.3%	0.29	46% / 88.5% / 60.5%	
Easy-IPC-Grid (30)	14.1	48.5	0.52	83.9% / 92.9% / 88.1%	0.30	100% / 96.2% / 98%	0.31	83.9% / 92.9% / 88.1%	
Ferry (30)	13.8	18.1	0.38	57.6% / 67.9% / 62.3%	0.13	100% / 78.6% / 88%	0.15	57.6% / 67.9% / 62.3%	
Logistics (30)	20.8	24.0	0.70	100% / 13% / 23.1%	0.11	100% / 91.3% / 95.5%	0.13	100% / 13% / 23.1%	
Satellite (20)	25.7	60.8	3.45	75% / 75% / 75%	0.44	45.5% / 66.7% / 54.1%	0.50	100% / 26.7% / 42.1%	
Sokoban (20)	24	76.5	3.77	10.5% / 42.9% / 16.9%	1.33	5.4% / 14.3% / 7.8%	1.45	35.5% / 78.6% / 48.9%	
Zeno-Travel (15)	12.2	38.7	0.71	77.8% / 50% / 60.9%	0.14	82.4% / 100% / 90.3%	0.18	77.8% / 50% / 60.9%	

Table 8. Experimental Results for Detecting Sub-Optimal Action Steps Combining the Techniques Analysing Plan Execution Deviation and Predicting Upcoming Actions via Landmarks (1)

Plan Optimality Monitoring (Analysing Plan Execution Deviation and Predicting Upcoming Actions via Landmarks)									
			h_{adjsum}		$h_{adjsum2}$		$h_{adjsum2M}$		
Domain	$\|O\|$	$\|\mathcal{L}\|$	**Time**	**PPV / TPR / F1**	**Time**	**PPV / TPR / F1**	**Time**	**PPV / TPR / F1**	
Driver-Log (20)	20.1	53.6	0.71	100% / 77.7% / 87.5%	0.68	100% / 94.4% / 97.1%	1.33	100% / 100% / 100%	
Depots (30)	16.7	64.7	1.34	71.8% / 88.4% / 79.3%	1.22	81.2% / 100% / 89.6%	2.15	75.6% / 93.3% / 83.5%	
Easy-IPC-Grid (30)	14.1	48.5	0.81	100% / 96.1% / 98%	0.77	100% / 100% / 100%	0.98	100% / 75% / 85.7%	
Ferry (30)	13.8	18.1	0.23	88% / 78.5% / 83.1%	0.18	88% / 78.5% / 83.1%	0.34	80% / 42.9% / 55.8%	
Logistics (30)	20.8	24.0	0.47	100% / 85.7% / 92.3%	0.35	100% / 91.3% / 95.4%	0.89	100% / 91.3% / 95.4%	
Satellite (20)	25.7	60.8	5.41	100% / 26.6% / 42.1%	4.35	87.5% / 46.6% / 60.8%	9.58	88.8% / 53.3% / 66.6%	
Sokoban (20)	24.0	76.5	3.45	64.7% / 78.6% / 71.0%	2.26	80.0% / 57.1% / 66.7%	4.13	60.0% / 64.3% / 62.1%	
Zeno-Travel (15)	12.2	38.7	1.07	87.5% / 50% / 63.6%	0.86	100% / 92.8% / 96.2%	1.52	100% / 85.7% / 92.3%	

Table 9. Experimental Results for Detecting Sub-Optimal Action Steps Combining the Techniques Analysing Plan Execution Deviation and Predicting Upcoming Actions via Landmarks (2)

Plan Optimality Monitoring (Analysing Plan Execution Deviation and Predicting Upcoming Actions via Landmarks)									
			h_{combo}		h_{ff}		h_{sum}		
Domain	$\|O\|$	$\|\mathcal{L}\|$	**Time**	**PPV / TPR / F1**	**Time**	**PPV / TPR / F1**	**Time**	**PPV / TPR / F1**	
Driver-Log (20)	20.1	53.6	1.38	100% / 77.7% / 87.5%	0.74	100% / 94.4% / 97.1%	0.85	100% / 77.7% / 87.5%	
Depots (30)	16.7	64.7	2.46	71.4% / 96.1% / 81.9%	1.43	81.2% / 100% / 89.6%	1.39	71.8% / 88.4% / 79.3%	
Easy-IPC-Grid (30)	14.1	48.5	1.08	100% / 96.1% / 98%	0.86	100% / 100% 100%	0.79	100% / 96.1% / 98%	
Ferry (30)	13.8	18.1	0.36	88% / 78.5% / 83.1%	0.32	88% / 78.5% / 83.1%	0.19	88% / 78.5% / 83.1%	
Logistics (30)	20.8	24.0	1.11	100% / 85.7% / 92.3%	0.55	100% / 91.3% / 95.4%	0.43	100% / 85.7% / 92.3%	
Satellite (20)	25.7	60.8	9.81	100% / 40% / 57.1%	4.94	87.5% / 46.6% / 60.8%	4.53	100% / 26.6% / 42.1%	
Sokoban (20)	24.0	76.5	4.28	73.3% / 78.6% / 75.9%	2.22	80.0% / 57.1% / 66.7%	2.07	64.7% / 78.6% / 71.0%	
Zeno-Travel (15)	12.2	38.7	1.45	87.5% / 50% / 63.6%	0.99	100% / 92.8% / 96.2%	0.92	87.5% / 50% / 63.6%	

TPR, which means that it is accurate to detect the set of actions that do not contribute to achieve the monitored goal (true positive results). However, in general, for all domains, the technique that uses landmarks alone is not precise enough to detect sub-optimal actions, since it returns many false positive actions that do not contribute to achieve the monitored goal.

Tables 6 and 7 show performance results for our optimality monitoring approaches using domain-independent heuristics. As can be seen in comparison to Table 5, these results are superior to our prediction of upcoming landmarks technique in all evaluated domains and problems for all metrics. Some planning heuristics lead to more accurate and faster predictions than others in some planning domains. For instance, the FAST-FORWARD h_{ff} heuristic has near perfect results for PPV, TPR, and F1 for EASY-IPC-GRID, FERRY, LOGISTICS, and ZENO-TRAVEL, whereas the results are poor for SOKOBAN and SATELLITE. Since all heuristics are relatively cheap to compute, monitoring time is overall very fast and is, at most, 3.77 seconds. Thus, our heuristic approaches to detect plan deviation are generally better performing than predicting upcoming landmarks.

Tables 8 and 9 show the results of our plan optimality monitoring approach combining both previous techniques. The resulting approach achieves better results than each technique on its own. We can see that for the DRIVER-LOG and EASY-IPC-GRID domains, our plan optimality monitoring approach yields perfect results (100% for all metrics) using different heuristics, such as ADJUSTED-SUM2M ($h_{adjsum2M}$) and FAST-FORWARD (h_{ff}), respectively, along with our technique that predicts upcoming actions via landmarks. Apart from the SATELLITE domain that under-performs for all metrics, our approach is near-perfect monitoring optimality at low runtime in all other planning domains, yielding very good results with different heuristics. We note that the poor results with respect to the SATELLITE domain are related to estimated distance provided by the heuristics. We analysed the output of our approach over the problems instances of the SATELLITE domain, and we observed that all heuristics (for most problem instances) do not detect when the observed actions deviate to achieve the monitored goal, and such issue happens because the heuristics are inaccurate for the problem instances of this particular domain. To overcome this issue, we intend to use more modern domain-independent planning heuristics and then evaluate our approach not only over this domain but also in all domains we used in our experiments. We also note that some heuristics outperform others for the same domain—for instance, the ADJUSTED-SUM2M heuristic underperforms against others for the FERRY domain. In summary, our approach yields very good results detecting sub-optimal plan steps in deterministic planning domains, with most domains having high *F1-scores* from perfect and near-perfect accuracy (depending on the heuristic), whereas only one domain (i.e., SATELLITE) has relatively inaccurate results. We also note that the combination of our techniques yield better results than using the techniques separately, over-performing for all metrics, as we show in Tables 5, 6, and 7.

5.5 Commitment Abandonment Detection Experiments

We evaluated our approach to detect commitment abandonment using the same metrics as before: *Precision* (PPV), *Recall* (TPR), and the *F1-score* (F1). Here, true positive results represent the number of plans that actually did abandon their expected commitments that our approach has detected correctly. False positive results represent the number of plans that actually eventually achieved the commitment consequent that our approach has detected as having abandoned the commitment. False negative results represent the number of plans that would not eventually reach the commitment consequent that our approach has not detected as abandonment.

Table 10 shows the experimental results of our commitment abandonment approach over the selected domains using the heuristics that yield best results to detect sub-optimal action steps. Each row details results for a different domain showing averages for the number of observations $|O|$ across problem instances, monitoring time (in seconds), *Precision* (PPV), *Recall* (TPR), and

Table 10. Experimental Results for Detecting Commitment Abandonment

Domain	Heuristic	$\|O\|$	Time	PPV θ (0% / 5% / 10%)	TPR θ (0% / 5% / 10%)	F1 θ (0% / 5% / 10%)
Driver-Log (30)	$h_{adjsum2M}$	20.0	0.83	100% / 100% / 100%	100% / 100% / 100%	100% / 100% / 100%
Depots (30)	$h_{adjsum2}$	18.6	1.79	100% / 100% / 100%	100% / 100% / 80.0%	100% / 100% / 88.8%
Easy-IPC-Grid (30)	h_{ff}	17.3	0.95	100% / 100% / 100%	100% / 100% / 100%	100% / 100% / 100%
Ferry (30)	$h_{adjsum2}$	13.5	0.38	100% / 100% / 100%	100% / 80.0% / 80.0%	100% / 88.8% / 88.8%
Logistics (30)	$h_{adjsum2}$	21.0	0.56	100% / 100% / 100%	100% / 100% / 100%	100% / 100% / 100%
Satellite (30)	$h_{adjsum2M}$	23.5	5.4	80% / 100% / 100%	80% / 60% / 60%	80% / 75% / 75%
Sokoban (30)	h_{combo}	22.8	5.2	83.3% / 100% / 100%	100% / 60% / 60%	90.9% / 75% / 75%
Zeno-Travel (30)	$h_{adjsum2}$	10.0	1.1	100% / 100% / 100%	80% / 80% / 80%	88.8% / 88.8% / 88.8%

F1-score (F1). The average number of observations ($|O|$), ranging between 10.0 and 23.5, indicate that all plans we analyse are non-trivial in complexity.

For the Driver-Log, Easy-IPC-Grid, and Logistics domains, our approach yields perfect predictions to detect commitment abandonment. Apart from the domains Satellite and Sokoban, that yield poor results (for threshold values 5% and 10%), for other domains we have near perfect prediction for detecting commitment abandonment. The results for Satellite and Sokoban are not as good as for other domains because the commitment abandonment detection is related to our plan optimality monitoring approach, which has not as good results for detecting sub-optimal action steps for the same domains, as we show in Tables 8 and 9. Thus, we can conclude that by using the detection of sub-optimal action steps, it is possible to accurately identify commitment abandonment in planning domains.

6 RELATED WORK

To the best of our knowledge, the most recent approach to monitor plan optimality was developed by Fritz and McIlraith [9]. This work formalises the problem of monitoring plan optimality by using situation calculus, a logical formalism to specify and reason about dynamical systems. Fritz and McIlraith seek to determine whether an execution follows an optimal plan but—unlike our work—do not seek to determine which actions are responsible for deviation from an optimal plan. Prior to that, Geib [10] and Geib and Goldman [11] developed a formal model of goal and plan abandonment detection. This formal model is based on plan libraries and estimates the probability that a set of observed actions in a sequence contributes to the goal being monitored. Unlike our work, which requires no prior knowledge of an agent's plan library, they assume knowledge about possible plan decompositions (i.e., a know-how of all plans to achieve a set of goals) available to each observed agent.

Siddiqui and Haslum [39] develop an approach that aims to detect deorderable (unordered) blocks of partial plans. According to the authors, a partial ordering implies that whenever two sub-plans are unordered, every interleaving steps from the two forms a valid execution. In this work, Siddiqui and Haslum propose a notion of partial ordering that divides the plan into blocks such that the action steps in a block may not be interleaved with action steps outside the block, but unordered blocks can be executed in any sequence. The authors argue that this approach can be used, for example, to break plans into sub-plans for distributed executions.

Proposed by Kafalı et al. [16], Gosu is an algorithm that uses commitments to regulate agents' interactions in an environment for achieving their goals. By using commitments as contractual agreements, this algorithm allows an agent to decide whether it can achieve its goals for a given set of commitments and the current state. Gosu does not use any planning approach to reason about agents' goals; it uses a depth-first search algorithm.

Kafalı and Yolum [17] propose a monitoring approach called PISAGOR that can determine whether a set of business interactions are progressing as expected in an e-commerce system. These business interactions are represented as commitments with deadlines. The authors also propose a set of operational rules for the observed agent to create expectations based on its commitments. Thus, PISAGOR monitors and detects whether the observed interaction is progressing well, and therefore it identifies what the problem is during the interactions.

Our technique to predict upcoming actions via landmarks (Section 3.3) is inspired by the landmark-based heuristics developed in Pereira and Meneguzzi [27] and Pereira et al. [30]. These authors developed two goal recognition heuristics that rely on planning landmarks, showing that it is possible to accurately recognise goals using just the concept of landmarks.

Pozanco et al. [34] develop an approach for counter-planning that combines the use of goal recognition, landmarks, and automated planning. The aim of this work is blocking the opponent's goal achievement by recognising the opponent's goal, landmark extraction to identify sub-goals that can be used to block goal achievement, and automated planning to generate plans that prevent goal achievement.

Criado [6] develops a multi-agent system approach to monitor and check norm compliance with limited resources, assuming incomplete information about the performed actions and the state of the world. This approach uses propositional logic to formalise the properties (state) of the world and norms, and action definitions with preconditions and effects to formalise how the agents act in the environment. For experiments and evaluation, this approach uses planning domains and problems (some of which we used in this work) also used in Ramírez and Geffner [35].

Meneguzzi et al. [22] develop a Hierarchical-Planning formalisation of the interplay between commitments and goals, allowing agents with a global view of the agent society to decide offline whether certain goals can indeed be accomplished either by individual agents or by complex networks of commitments. Such networks allow agents to delegate part of their goals and cooperate towards achieving interdependent individual goals. By having a global view of the planning problem and the delegation, Meneguzzi et al. assume that agents that do commit to each other achieve delegated goals to decide, offline, whether the goals can be achieved. Our work, by contrast, takes an online individualistic view of each agent as it delegates goals to other agents. Here, individual agents monitor debtors while they act, and when such debtors fail to achieve the consequent of a commitment, they may take appropriate action to ensure that they find alternative ways to achieve delegated goals.

Most recently, Telang et al. [43] formalised and proved a number of properties of the combined goal and commitment protocols implemented by Meneguzzi et al. [22, 23] and Telang et al. [42]. Such formalisation takes a similar offline global view of goals and commitments in a larger society, and focuses on similarly global properties of the *possible* plans that can achieve individual agent goals and fulfil their commitments, whereas our work reasons about the actual plans used by the agents as they operate in the environment.

7 CONCLUSION

In this article, we have developed planning-based approaches for monitoring plan optimality and detecting commitment abandonment. Our approaches use planning techniques but do not require executing a full planning algorithm by exploiting landmarks and domain-independent heuristics. These provide useful information that can be used in planning-based approaches for recognising goals and plans. Finally, our plan optimality approach can also be used to repair sub-optimal plans, such as by detecting which parts of the plan are sub-optimal and then improving it.

As we show in experiments and evaluation, our approaches yield very accurate results for detecting sub-optimal plan steps and commitment abandonment, dealing with realistic well-known

deterministic planning domains. Thus, our approaches can provide timely accurate estimates of when an agent fails to accomplish a delegated commitment, allowing creditors (i.e., the agents relying on the commitment being achieved) to decide on which action to take to achieve their individual goals. Such actions might consist of reminding the debtor or sanctioning unreliable debtors (when applicable) and quickly arrange for an alternative agent to accomplish the desired delegated goal. Such techniques thus provide a major contribution to the design of autonomous agent societies that rely on networks of commitments to achieve societal goals [4, 23, 41], allowing agents to quickly compute which agents are accountable [3] when networks of commitments fail.

7.1 Limitations and Future Work

The work described in this article makes several simplifying assumptions, but these assumptions impact on its applicability to some domains. Therefore, one strand of future work involves relaxing these assumptions. One such assumption is of full observability of the environment during the monitoring process. An approach to dealing with partial observability involves using an automated planner to fill in missing observations. Related to this is our assumption of uniform action costs, which we can relax through the use of cost-based planning heuristics [18, 33]. Combining these two will introduce a new assumption, namely that the (least cost) plan identified by the planner is the one utilised by the agent. Evaluating whether this new assumption is appropriate is another important strand of work. Finally, to deal with interleaving plans and multiple goals, we will use other landmark extraction algorithms [19, 36, 45] and investigate the effectiveness of more modern planning heuristics [32, 38, 40].

REFERENCES

[1] Fahiem Bacchus. 2001. The AIPS'00 planning competition. *AI Magazine* 22, 3 (2001), 47–56.
[2] Matteo Baldoni, Cristina Baroglio, Federico Capuzzimati, and Roberto Micalizio. 2015. Programming with commitments and goals in JaCaMo+. In *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'15)*. 1705–1706.
[3] Matteo Baldoni, Cristina Baroglio, Katherine M. May, Roberto Micalizio, and Stefano Tedeschi 0001. 2018. An information model for computing accountabilities. In *AI*IA 2018—Advances in Artificial Intelligence*. Lecture Notes in Computer Science, Vol. 11298. Springer, 30–44.
[4] Matteo Baldoni, Cristina Baroglio, and Roberto Micalizio. 2015. Social continual planning in open multiagent systems: A first study. In *Principles and Practice of Multi-Agent Systems (PRIMA)*. Springer, 575–584.
[5] Blai Bonet and Hector Geffner. 2001. Planning as heuristic search. *Journal of Artificial Intelligence Research* 129, 1–2 (2001), 5–33.
[6] Natalia Criado. 2018. Resource-bounded norm monitoring in multi-agent systems. *Journal of Artificial Intelligence Research* 62, 1–2 (2018), 153–192.
[7] Carmel Domshlak, Michael Katz, and Sagi Lefler. 2012. Landmark-enhanced abstraction heuristics. *Artificial Intelligence* 189 (2012), 48–68.
[8] Maria Fox and Derek Long. 2002. The Third International Planning Competition: Temporal and metric planning. In *Proceedings of the 6th International Conference on Artificial Intelligence Planning Systems*. 333.
[9] Christian Fritz and Sheila A. McIlraith. 2007. Monitoring plan optimality during execution. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*. 144–151.
[10] Christopher W. Geib. 2002. Problems with intent recognition for elder care. In *Proceedings of the 18th AAAI Conference on Artificial Intelligence (AAAI'02)*. 13–17.
[11] Christopher W. Geib and Robert P. Goldman. 2003. Recognizing plan/goal abandonment. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*.1515–1517.
[12] Malik Ghallab, Dana S. Nau, and Paolo Traverso. 2004. *Automated Planning: Theory and Practice*.Elsevier.
[13] Malte Helmert. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26, 1 (2006), 191–246.
[14] Jörg Hoffmann and Bernhard Nebel. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14, 1 (2001), 253–302.
[15] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research* 22, 1 (2004), 215–278.

[16] Özgür Kafalı, Akın Günay, and Pınar Yolum. 2014. GOSU: Computing GOal SUpport with commitments in multiagent systems. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI'14)*. 477–482.
[17] Özgür Kafalı and Pınar Yolum. 2016. PISAGOR: A proactive software agent for monitoring interactions. *Knowledge and Information Systems* 47, 1 (2016), 215–239.
[18] Emil Keyder and Héctor Geffner. 2008. Heuristics for planning with action costs revisited. In *Proceedings of the 18th European Conference on Artificial Intelligence (ECAI'08)*. 588–592.
[19] Emil Keyder, Silvia Richter, and Malte Helmert. 2010. Sound and complete landmarks for and/or graphs. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI'10)*. 335–340.
[20] Nir Lipovetzky and Hector Geffner. 2011. Searching for plans with carefully designed probes. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS'11)*.
[21] Derek Long, Henry A. Kautz, Bart Selman, Blai Bonet, Hector Geffner, Jana Koehler, Michael Brenner, et al. 2000. The AIPS-98 Planning Competition. *AI Magazine* 21, 2 (2000), 13–33.
[22] Felipe Meneguzzi, Mauricio C. Magnaguagno, Munindar P. Singh, Pankaj R. Telang, and Neil Yorke-Smith. 2018. GoCo: Planning expressive commitment protocols. *Autonomous Agents and Multi-Agent Systems* 32, 4 (2018), 459–502.
[23] Felipe Meneguzzi, Pankaj R. Telang, and Munindar P. Singh. 2013. A first-order formalization of commitments and goals for planning. In *Proceedings of the 27th Conference on Artificial Intelligence (AAAI'13)*. 697–703.
[24] XuanLong Nguyen and Subbarao Kambhampati. 2000. Extracting effective and admissible state space heuristics from the planning graph. In *Proceedings of the 17th National Conference on Artificial Intelligence and the 12th Conference on on Innovative Applications of Artificial Intelligence*. 798–805.
[25] XuanLong Nguyen, Subbarao Kambhampati, and Romeo Sanchez Nigenda. 2002. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence* 135, 1–2 (2002), 73–123.
[26] David Pattison and Derek Long. 2010. Domain independent goal recognition. In *STAIRS*. Frontiers in Artificial Intelligence and Applications, T. Agotnes (Ed.). IOS Press, 238–250.
[27] Ramon Fraga Pereira and Felipe Meneguzzi. 2016. Landmark-based plan recognition. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI'16)*.
[28] Ramon Fraga Pereira and Felipe Meneguzzi. 2017. Goal and plan recognition datasets using classical planning domains. *Zenodo*. Retrieved December 17, 2019 from DOI: https://doi.org/10.5281/zenodo.825878
[29] Ramon Fraga Pereira, Nir Oren, and Felipe Meneguzzi. 2017. Detecting commitment abandonment by monitoring sub-optimal steps during plan execution. In *Proceedings of the 16th Conference on Autonomous Agents and Multiagent Systems (AAMAS'17)*. 1685–1687.
[30] Ramon Fraga Pereira, Nir Oren, and Felipe Meneguzzi. 2017. Landmark-based heuristics for goal recognition. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*.
[31] Ramon Fraga Pereira, Nir Oren, and Felipe Meneguzzi. 2017. Monitoring plan optimality using landmarks and domain-independent heuristics. In *Proceedings of the AAAI 2017 Workshop on Plan, Activity, and Intent Recognition*.
[32] Florian Pommerening, Malte Helmert, and Blai Bonet. 2017. Higher-dimensional potential heuristics for optimal classical planning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*. 3636–3643.
[33] Florian Pommerening, Gabriele Röger, Malte Helmert, and Blai Bonet. 2014. LP-based heuristics for cost-optimal planning. In *Proceedings of the 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*.
[34] Alberto Pozanco, Yolanda Escudero, Susana Fernandez, and Daniel Borrajo. 2018. Counterplanning using goal recognition and landmarks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'18)*. 4808–4814.
[35] Miquel Ramírez and Hector Geffner. 2009. Plan recognition as planning. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*. 1778–1783.
[36] Silvia Richter, Malte Helmert, and Matthias Westphal. 2008. Landmarks revisited. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI'08)*. 975–982.
[37] Silvia Richter and Matthias Westphal. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research* 39, 1 (2010), 127–177.
[38] Jendrik Seipp, Florian Pommerening, and Malte Helmert. 2015. New optimization functions for potential heuristics. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling (ICAPS'15)*. 193–201.
[39] Fazlul Hasan Siddiqui and Patrik Haslum. 2012. Block-structured plan deordering. In *Proceedings of the 25th Australasian Joint Conference on Advances in Artificial Intelligence*. 803–814.
[40] Silvan Sievers, Martin Wehrle, and Malte Helmert. 2016. An analysis of merge strategies for merge-and-shrink heuristics. In *Proceedings of the 26th International Conference on Automated Planning and Scheduling (ICAPS'16)*. 294–298.
[41] Munindar P. Singh. 2008. Semantical considerations on dialectical and practical commitments. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI'08)*. 176–181.

[42] Pankaj Telang, Felipe Meneguzzi, and Munindar Singh. 2013. Hierarchical planning about goals and commitments. In *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'13)*. 877–884.
[43] Pankaj Telang, Munindar P. Singh, and Neil Yorke-Smith. 2019. A coupled operational semantics for goals and commitments. *Journal of Artificial Intelligence Research* 65 (2019), 31–85.
[44] Vincent Vidal. 2004. A lookahead strategy for heuristic search planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS'04)*. 150–160.
[45] Lin Zhu and Robert Givan. 2003. Landmark extraction via planning graph propagation. In *Proceedings of theDoctoral Consortium at the International Conference on Automated Planning and Scheduling (ICAPS'03)*.

Received March 2019; revised September 2019; accepted November 2019