Adaptive Goal Recognition Using Process Mining Techniques

Zihang Su[a,*], Artem Polyvyanyy[a], Nir Lipovetzky[a], Sebastian Sardiña[b], Nick van Beest[c]

[a]*The University of Melbourne, 700 Swanston St., Carlton, 3053, VIC, Australia*
[b]*Royal Melbourne Institute of Technology, 124 La Trobe St., Melbourne, 3000, VIC, Australia*
[c]*Data61 | CSIRO, 41 Boggo Road, Dutton Park, 4102, QLD, Australia*

Abstract

Goal Recognition (GR) is a research problem that studies ways to infer the goal of an intelligent agent based on its observed behavior and knowledge of the environment in which the agent operates. A common assumption of GR is that the environment is static. However, in many real-world scenarios, for example, recognizing customers' preferences, it is necessary to recognize the goals of multiple agents or multiple goals of a single agent over an extended period. Therefore, it is reasonable to expect the environment to change throughout a series of goal recognition tasks. This paper presents three process mining-based solutions to the problem of *adaptive* GR in a changing environment implemented as different control strategies of a system for solving standard GR problems. As a standard GR system that gets controlled, we use the system grounded in process mining techniques, as it can adjust its internal GR mechanisms based on data collected while observing the operating agents. We evaluated our control strategies over synthetic and real-world datasets. The synthetic datasets were generated using the extended version of the Goal Recognition Amidst Changing Environments (GRACE) tool. The datasets account for different types of changes and drifts in the environment. The evaluation results demonstrate a trade-off between the GR performance over time and the effort invested in adaptations of the GR mechanisms of the system, showing that few well-planned adaptations can lead to a consistently high GR performance.

Keywords: Adaptive goal recognition, changing environment, process mining, GRACE

1. Introduction

Goal recognition (GR) aims to develop systems capable of identifying the goal, or intention, of an intelligent agent based on its observed behavior in the environment. Such an observed behavior is often captured in a *trace*, a sequence of observed actions performed by the agent. Being a key aspect of Theory of Mind [1, 2], the GR problem has been significantly studied in Artificial Intelligence [3, 4], with the first formalization often attributed to Kautz and Allen [5]. As computer-based systems become increasingly more autonomous, the GR problem has recently received more attention [4]. Indeed, the problem is central to the realization of truly intelligent situated systems in numerous applications, including adversarial reasoning for games

*Corresponding author
Email address: `zihangs@student.unimelb.edu.au` (Zihang Su)

and the military [6, 7], dialogue systems [8, 9, 10], behavior understanding in smart homes [11], wheelchair movement support [12], trajectory/maneuver prediction [13, 14, 15], and human-computer collaboration [16].

Existing GR techniques, such as the plan library-based approaches [5, 17, 18], the planning-based approaches [19, 20], and the process mining (PM-)based approach [21, 22], propose algorithms for solving a "single-shot" GR problem; that is, to correctly infer the most likely goal the observed agent is aiming to achieve assuming that the environment is *static*. However, in real-world scenarios like the ones mentioned above, there is a need to solve multiple GR problems over an *extended* period, *during which the environment may change*. Importantly, when the environment changes, the behavior of the agent for achieving its goals will generally also change [23]. For example, in a navigational scenario, when new road blocks appear or new shortcuts are created, intelligent agents may take different routes to get to their destination locations. In turn, a smart home system performs goal recognition on the households to support their daily activities, and their behaviors may vary depending on the season. While people use heaters in winter to get comfort temperature, they tend to use cooling systems in summer instead (for the same goal). The behavior of the households is also expected to change upon changes in the house configuration (e.g., changing the location of the TV to another room). This phenomenon is also observed in business processes when the behavior of process participants changes to address new regulations, compliance rules, and innovative ways of doing business [24]. In all those cases, we would like a GR system able to first recognize the changes in (observed) behavior, and to then adapt accordingly so as to maintain its recognition performance.

We refer to the problem of solving multiple GR problems over a time interval during which the environment in which the agent operates may change as the *adaptive GR problem*, and provide a formal definition of this problem. To achieve high recognition accuracy in solving the adaptive GR problem, the GR system needs to detect environmental changes or changes in the agent's behavior and determine the optimal timing for updating its knowledge models based on new evidence. We illustrate the intrinsic trade-off between the cost associated with updating knowledge models and the resulting GR accuracy, which can be formulated as an optimization problem. In this paper, we propose three solutions to address this problem and present the trade-off by utilizing synthetic and real-world experimental data. These solutions are designed as open- and closed-loop control strategies over standard single-shot GR systems. In this work, we use the GR system grounded in process mining techniques [21, 22] as one such concrete standard GR system, because it can automatically learn and relearn its recognition mechanisms based on data generated while observing the behavior of the agent of interest. Specifically, the standard GR system is controlled based on its performance to request, for instance, to relearn its knowledge base if the recognition accuracy drops.

As research of the adaptive GR problem requires experimental datasets for comparing candidate solutions, we use and extend the Goal Recognition Amidst Changing Environments (GRACE) tool [25] to generate synthetic instances of the new problem. For the seed environments in which observed agents operate, we use the static IPC domains[1] widely used in classical GR. These environments are provided as input to GRACE, which subsequently generates problem instances in which the seed environments change over time according to the parameters supplied to the tool. The GRACE tool uses two aspects to characterize a change in an environment. Firstly, a change is characterized based on which components of the environment change.

[1] https://github.com/pucrs-automated-planning/goal-plan-recognition-dataset

For example, such components are the initial and goal states of the agents and actions the agents can perform in the environment. Secondly, by interpreting an environment as a *signal* and, consequently, a change in the original environment as a change in the original signal, we characterize changes in the environment based on the different types of concept drift [24]. For example, a change in an environment can be sudden or can be such that it progresses gradually over time. The GRACE tool uses random modifications to generate environment changes, so the modified environment may not differ significantly, if at all, from the original. Therefore, we introduce a measure that quantifies the significance of a change and extend GRACE to use this measure to ensure the generated problem instances are defined over the environment that changes significantly over time. In addition, we formulate instances of adaptive GR problems using real-world datasets from the health domain. We rely on these synthetic and real-world problem instances to evaluate how well our GR systems solve adaptive GR problems. The results show that, compared to the single-shot GR system, the new systems that control the single-shot system, over time, recognize goals more accurately.

Concretely, this paper makes the following contributions:

- It formally defines the adaptive GR problem, which can be seen as an optimization problem that balances the cost of adapting knowledge models with the accuracy of GR.
- It presents a measure of the significance of a change in the environment and extends GRACE to use this measure for generating adaptive GR problem instances with significant changes in the environment.
- It presents a synthetic dataset of adaptive GR problem instances generated by the extended GRACE tool using IPC domains as seed environments.
- It presents and evaluates, using our publicly available implementation and the synthetic and real-world datasets, three adaptive GR systems implemented as control mechanisms over a standard GR system grounded in process mining techniques. The evaluation results confirm the effectiveness of these GR systems for solving adaptive GR problems.

The next section motivates and presents the adaptive GR problem. Then, Section 3 introduces a GR system grounded in process mining techniques that works in static environments, while Section 4 presents its extension that relaxes this assumption. Section 5 summarizes the results of evaluating our GR system over synthetic and real-world instances of the adaptive GR problems. We discuss related work in Section 6. Finally, Sections 7 and 8 discuss limitations and directions for future work and provide concluding remarks.

2. Adaptive Goal Recognition

An *adaptive* algorithm aims to improve its performance based on the inputs and changing conditions. Such an algorithm is useful, for instance, when the problem it solves is defined over a dynamic or uncertain environment. Let $\mathcal{F}$ be the universe of all possible *facts* and let $\mathcal{O} = \mathcal{F} \times \mathbb{N}_0$ be the universe of all possible (timestamped) facts, or *observations*, where $(f, t) \in \mathcal{O}$ denotes an observation of fact f at time t. The GR problem can, in general, be defined as follows.

Definition 1 (Probabilistic goal recognition). Given a triple $\langle \mathcal{G}, \mathcal{K}, O \rangle$, where $\mathcal{G}$ is a set of candidate *goals*, $\mathcal{K}$ is the available *knowledge* on an agent and the environment, and $O \subseteq \mathcal{O}$ are *observations* of the *agent* operating in the environment, the *probabilistic goal recognition* problem involves obtaining a posterior probability distribution over $\mathcal{G}$ based on $\mathcal{K}$ and O that describes the likelihood of the agent achieving the different goals.

Different instantiations of this definition of the GR problem have been proposed in the literature. They differ in how the elements of the problem and its solution are represented. In the approach by Ramirez and Geffner [26], the knowledge is represented by a domain theory and the information on when the agent's actions are applicable in the environment and how they transform it, while in the technique by Polyvyanyy et al. [21], the knowledge is given as collections of historical action sequences toward goals. The observations usually capture a time-ordered sequence of *actions* performed by the agent thus far. Finally, the inferred goals can be identified by a set $G \subseteq \mathcal{G}$, for instance, as a subset of the most likely goals of the agent.

Let $\mathcal{P}$, $\mathcal{S}$, and $\mathcal{T}$ denote the universes of all GR problems, solutions to these problems (for instance, probability distributions over candidate goals), and (true) goals, respectively. A goal recognition algorithm α maps problems to solutions, that is, $\alpha : \mathcal{P} \rightarrow \mathcal{S}$, where $\alpha(p)$ is the solution to problem $p \in \mathcal{P}$ by algorithm α. Given a solution $\alpha(p)$ to problem p and the *true* goal $g \in \mathcal{T}$ of the agent from p, one can assess the quality of the solution using standard measures such as precision, recall, and accuracy. Given a sequence X of length n, by $X[i]$, $i \in [0 \mathinner{..} n]$, we denote the prefix of X of length i. By X_i, we denote the element of sequence X at position i.

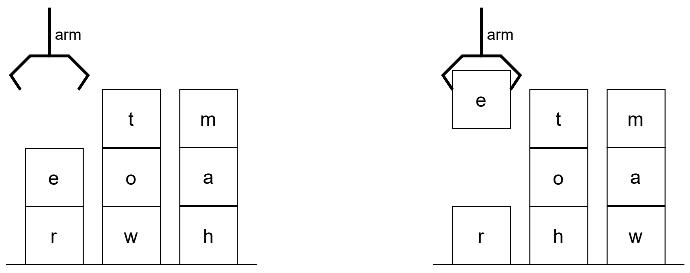
Then, the adaptive GR problem is defined as follows.

Definition 2 (Adaptive goal recognition). A triple $\langle P, S, T \rangle$, where $P \in \mathcal{P}^*$, $S \in \mathcal{S}^*$, $T \in \mathcal{T}^*$, $|P| = n$, and $|S| = n - 1 = |T|$, is an *adaptive goal recognition* problem of size n that consists in solving GR problem P_n if either (i) $n = 1$ or (ii) $n > 1$, S_i is a solution to problem P_i, T_i is the true goal of the agent from problem P_i, $i \in [1 \mathinner{..} n-1]$, and $\langle P[n-1], S[n-2], T[n-2] \rangle$ is an adaptive GR problem of size $n - 1$.

The adaptive GR problem reduces to the conventional GR problem (Definition 1) when $n = 1$ and is a generalization of the latter. The adaptive GR problem can also be captured as a special GR problem with the candidate goals and observations of P_n and the knowledge given by P, S, and T. However, the explicit inductive structure of Definition 2 suggests that each subsequent goal recognition can happen under new conditions and, thus, it is possible that $P_i = P_j$ but $S_i \neq S_j$, or $S_i = S_j$ but $P_i \neq P_j$. The former situation can occur when a solution to the same problem learns from its previous solution and adapts it, aiming to improve the quality of the new solution. The latter situation may arise when different knowledge of the environment or observations of the agent confirm the same inferred goal.

An algorithm for solving adaptive GR problems can use available information on the quality of previous GR solutions to improve its performance. Interestingly, we do not require all the solutions in S to stem from the same algorithm. However, in practice, one can often expect that S_{n-1} is obtained by the same algorithm that is used to solve P_n when solving the adaptive problem of size $n - 1$ defined by the prefixes of P, S, and T. Finally, the performance of a GR algorithm that tackles an adaptive GR problem can be assessed over time, that is, over all the induced problems of sizes from one to n.

Figure 1 shows two environments in the blocks-world domain. An environment in this domain consists of a set of blocks, each either lying on the table or stacked on top of another block, and an arm that can either be empty or hold a single block. For example, in Fig. 1a, the arm is empty, blocks *r*, *w*, and *h* are on the table, block *e* is stacked on top of block *r*, block *o* is stacked on top of block *w*, block *t* is stacked on top of block *o*, block *a* is stacked on top of block *h*, and block *m* is stacked on top of block *a*. When empty, the arm, controlled by an agent, can pick up single block *x* from the table, using action (`pick-up` *x*), or unstack *x* from block *y*, using action (`unstack` *x y*), if no other blocks are stacked on top of block *x*. After picking up



(a) Environment 0 (b) Environment 1

Figure 1: Two environments in the blocks-world domain.

a block, the arm is no longer empty, as it now holds the block, and cannot pick up other blocks. The arm can put block x it is holding back onto the table, using action `(put-down` x`)`, or stack it on top of block y, using action `(stack` x y`)`.

Let the environment in Fig. 1a be the initial environment from which an agent starts to control the arm and attempts to achieve one of two candidate goals: G_1 or G_2. Goal G_1 is to build the blocks that form the word "tower" where there are no blocks on t, block r is on the table, t stacks on o, o stacks on w, w stacks on e, and e stacks on r. Goal G_2 is to build the blocks and form the word "mother" where there are no blocks on m, block r is on the table, m stacks on o, o stacks on t, t stacks on h, h stacks on e, and e stacks on r. The actions of the agent are the ability to use the arm to pick up and put down blocks. The knowledge about the environment and agent determines which blocks the arm can pick up and where it can put the block it holds.

One can capture the described situation as the probabilistic goal recognition problem $\langle \{G_1, G_2\}, \mathcal{K}, O \rangle$, cf. Definition 1, where $\mathcal{K}$ describes the environment in Fig. 1a and the actions the agent can perform in different circumstances, and O is an observation of the agent's historical actions. For example, one can use the sequence $\langle$`(unstack t o)`, `(put-down t)`, `(unstack o w)`, `(put-down o)`, `(pick-up w)`, `(stack w e)`$\rangle$ to specify that six consecutive actions performed by the agent were observed. A solution to this problem can state, for instance, that the agent aims for G_1 with the probability of 0.7 and G_2 with the probability of 0.3.

In practice, an agent may need to achieve the same goal repeatedly, for example, to practice or learn different operation strategies or to accomplish multiple instances of the same task. In general, the environment and the behavior of the agent may change over time. For example, the initial state of the environment in which the agent attempts to achieve the goals can change from the one shown in Fig. 1a to the one depicted in Fig. 1b. Other changes that may affect the agent's performance include changes in the candidate goals, changes in the knowledge about the principles that govern the evolution of the environment, and the evolution of the skills of the agent to operate in the environment [25]. Such ongoing changes to the environment can be referred to as *concept drifts*. Figure 2 visualizes an example gradual drift of the initial state of the environment from environment 0 to environment 1 generated by the GRACE tool [25] that happens between time steps 50 and 100. In this type of drift, the initial state defined by environment 0 is observed up to and including time step 67. Subsequently, between time steps 67 and 88, the initial state gradually changes from environment 0 to environment 1, with environment 1 appearing more frequently over time until from time step 88 onward the initial state is entirely defined by environment 1.

A data-driven GR algorithm, like the PM-based GR system [21, 22], often has a training stage. During training, such an algorithm can use historical observations and feedback from

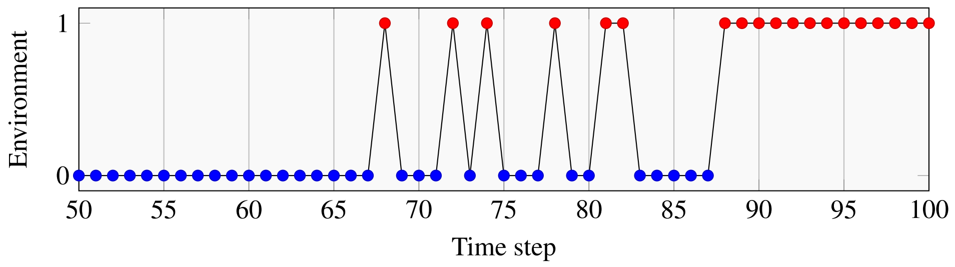


Figure 2: An example gradual drift from environment 0 to environment 1 generated by GRACE.

existing GR solutions to learn how to carry out future goal recognition practices best. Such training is often associated with a cost, for instance, computational resources and time spent for training. While there is a desire to minimize these costs, frequent trainings may be necessary when the environment, goals, or the agent's behavior drifts. These conflicting aims call for exploring different strategies to minimize training costs while maximizing GR quality over time.

Given two sequences X and Y, $X \circ Y$ denotes their concatenation. Let $P \in \mathcal{P}^*$ be a non-empty sequence of GR problems of length n. By $P(1)$, we denote the adaptive GR problem $\langle P[1], \langle\rangle, \langle\rangle\rangle$. Then, by $P(i)$, $i \in [2\,..\,n]$, we denote the adaptive GR problem $\langle P[i], S, T\rangle$, where $S = S' \circ \langle s\rangle$ and $T = T' \circ \langle g\rangle$, such that S' and T' are, respectively, the solutions and goals of $P(i{-}1)$ and s and g are, respectively, the solution of problem $P(i{-}1)$ and the true goal of the agent from problem $P(i-1)$. Then, an *adaptive GR strategy* is a solution to the optimization problem that maximizes the quality of solutions to GR problems while minimizing the cost invested in training the GR algorithm to obtain the solutions.

Definition 3 (Adaptive GR strategy). Given a GR algorithm $\alpha : \mathcal{P} \to \mathcal{S}$ and a non-empty sequence of GR problems $P \in \mathcal{P}^*$ of length n, an *adaptive GR strategy* is a set of positions I in P, such that for each position $i \in I$ algorithm α is trained on the solutions and true goals of $P(i)$ before it is applied to solve problems $\{P(k) \mid k \in [i\,..\,n]\}$, aiming to maximize an objective function that praises the GR solutions of high quality and condemns training of α of high costs.

To exemplify the trade-off between the performance of GR inference and the number of times the system is trained, Fig. 3 compares the performance of the conventional PM-based GR system and the same system controlled by the open-loop adaptive strategy, which regularly triggers the system to relearn its knowledge models based on which it carries out the inference. The inference is made for the two goals of building words "tower" and "mother" starting from one of the two environments shown in Fig. 1. The initial state of the environment for each time step is determined by the drift depicted in Fig. 2. In the figure, green boxes denote the accuracy of the adaptive GR system that relearns every ten time steps using the data collected from the ten most recent inferences. Vertical blue dashed lines in the figure denote the time steps when the system is retrained. In contrast, red crosses denote the accuracy of the conventional GR system that relies on the same knowledge models throughout the entire inference period. The adaptive GR system achieves an average accuracy of 0.90 over the period between time steps 50 and 100 while being trained four times, while the conventional GR system achieves an average accuracy of 0.75 without updates of its knowledge models, showing the value of the adaptive strategy.

Each time step in Figs. 2 and 3 defines an adaptive GR problem, cf. Definition 2, and reports the quality of its solution. Specifically, for a time step i, the corresponding adaptive GR problem is defined by $\langle P^i, S^i, T^i\rangle$, where P^i is the sequence of all GR problems requested to be solved up to, and including, time step i, S^i is the sequence of all solutions to the GR problems solved up to,

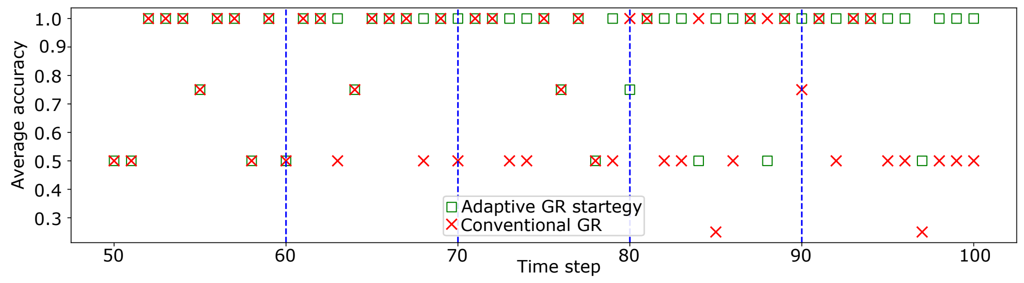


Figure 3: Accuracy of a conventional GR system and a GR system that implements an open-loop adaptive strategy.

but excluding, time step i, and T^i is the sequence of all true goals the agent achieved in the past up to, but excluding, time step i. The accuracy of the solutions in S^i with respect to the true goals from T^i is reported in Fig. 3. The GR problem P^i_j can be captured as triplet $\langle\{G_1, G_2\}, \mathcal{K}_j, O_j\rangle$, cf. Definition 1, where $\mathcal{K}_j$ is the knowledge model defined by the environment at time step j in Fig. 2 and O_j are the available observations of the agent in that environment.

3. Process Mining-Based Goal Recognition in Static Environments

A GR system grounded in process mining (PM) [27] techniques can solve single-shot GR problems [21, 22]. This GR system learns *process models* from historical observations of agents' actions that describe the behavior exhibited by the agents in the past to achieve their goals. It subsequently uses these models to recognize the goal of an agent by assessing the similarity between the currently observed actions of the agent and the learned models. Figure 4 shows the architecture of the PM-based GR system, including the data flow. It consists of three steps, which are *process discovery*, *conformance checking*, and *probability calculation*. Next, we use the example from Fig. 1a to describe the three steps of the PM-based GR approach in detail.

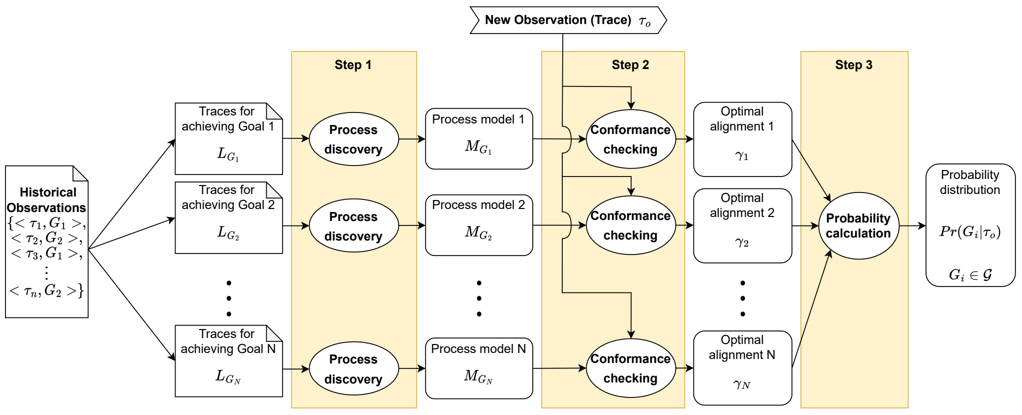

New Observation (Trace) τ_o
Step 1
Step 2
Step 3
Historical Observations
$\{<\tau_1, G_1>, <\tau_2, G_2>, <\tau_3, G_1>, \vdots <\tau_n, G_2>\}$
Traces for achieving Goal 1 L_{G_1}
Traces for achieving Goal 2 L_{G_2}
Traces for achieving Goal N L_{G_N}
Process discovery
Process discovery
Process discovery
Process model 1 M_{G_1}
Process model 2 M_{G_2}
Process model N M_{G_N}
Conformance checking
Conformance checking
Conformance checking
Optimal alignment 1 γ_1
Optimal alignment 2 γ_2
Optimal alignment N γ_N
Probability calculation
Probability distribution
$Pr(G_i|\tau_o)$
$G_i \in \mathcal{G}$

Figure 4: Architecture of the PM-based GR system.

Step 1: Process discovery is a technique that automatically constructs a process model from an *event log* given as a collection of traces [28]. The PM-based GR system uses a process discovery technique to learn process models, captured as *Petri nets* [29], each describing *sequences of* *actions* for achieving a specific goal. Note that a discovered model generalizes the traces in the event log used to construct it [30], thus describing yet unseen traces toward the goal. In the PM-based GR system, learning the process models is the prerequisite for performing goal recognition, and the system learns process models from historical observations of agents' actions and the goals they led to. Given a set of N candidate goals, denoted by $\mathcal{G}$, by $\langle \tau, G \rangle$, where $G \in \mathcal{G}$, we capture the goal G achieved by trace τ. Grouping the traces by the associated goals leads to multiple subsets of traces, where each subset can be seen as an event log, denoted by L_G, that only contains the traces toward goal G. Subsequently, using a process discovery algorithm, for instance, Directly Follows Miner [31], a process model is discovered from each resulting event log, where each constructed model M_G describes the behavior for achieving goal G.

τ_1	τ_2	τ_3	τ_4	τ_5
⟨(unstack t o),	⟨(unstack t o),	⟨(unstack t o),	⟨(unstack t o),	⟨(unstack t o),
(put-down t),	(stack t m),	(put-down t),	(put-down t),	(stack t m),
(unstack o w),	(unstack o w),	(unstack o w),	(unstack o w),	(unstack o w),
(put-down o),	(put-down o),	(stack o m),	(stack o t),	(stack o t),
(pick-up w),	(pick-up w),	(pick-up w),	(pick-up w),	(pick-up w),
(stack w e),	(stack w e),	(stack w e),	(stack w e),	(stack w e),
(pick-up o),	(pick-up o),	(unstack o m),	(unstack o t),	(unstack o t),
(stack o w),	(stack o w),	(stack o w),	(stack o w),	(stack o w),
(pick-up t),	(unstack t m),	(pick-up t),	(pick-up t),	(unstack t m),
(stack t o)⟩	(stack t o)⟩	(stack t o)⟩	(stack t o)⟩	(stack t o)⟩

Table 1: Event log L_{G_1} comprising five traces τ_1 to τ_5 for achieving goal G_1.

τ_6	τ_7	τ_8	τ_9	τ_{10}
⟨(unstack m a),	⟨(unstack m a),	⟨(unstack m a),	⟨(unstack m a),	⟨(unstack m a),
(put-down m),	(put-down m),	(put-down m),	(put-down m),	(put-down m),
(unstack a h),	(unstack a h),	(unstack a h),	(unstack a h),	(unstack a h),
(put-down a),	(put-down a),	(put-down a),	(put-down a),	(stack a e),
(pick-up h),	(pick-up h),	(pick-up h),	(pick-up a),	(unstack a e),
(stack h e),	(stack h e),	(stack h e),	(put-down a),	(put-down a),
(unstack t o),	(unstack t o),	(unstack t o),	(pick-up h),	(pick-up h),
(stack t h),	(stack t h),	(stack t h),	(stack h e),	(stack h e),
(unstack o w),	(unstack o w),	(unstack o w),	(unstack t o),	(unstack t o),
(stack o t),	(stack o t),	(stack o t),	(stack t h),	(stack t h),
(pick-up m),	(pick-up m),	(pick-up m),	(unstack o w),	(unstack o w),
(stack m o)⟩	(stack m o),	(stack m o),	(stack o t),	(stack o t),
	(pick-up a)⟩	(pick-up w)⟩	(pick-up m),	(pick-up m),
			(stack m o)⟩	(stack m o)⟩

Table 2: Event log L_{G_2} comprising five traces τ_6 to τ_{10} for achieving goal G_2.

Starting from the initial state shown in Fig. 1a, the agent controls the arm to achieve one of the two candidate goals: to form the word "tower" (G_1) and to form the word "mother" (G_2). Let us assume that we have already observed two distinct sets of traces (event logs), denoted by L_{G_1} and L_{G_2}, each containing five traces as presented in Tables 1 and 2. The GR system can subsequently learn two process models from L_{G_1} and L_{G_2}, namely M_{G_1} and M_{G_2}, represented graphically as Petri nets in Fig. 5 and Fig. 6, respectively.[2]

Step 2: Conformance checking techniques measure and describe the commonalities and discrepancies between a trace and a process model. The PM-based GR system uses the conformance checking technique of *optimal alignments* to compare a newly observed trace τ_o of

[2]In the figures, we abbreviate agent's actions as follows: `pu` = `pick-up`, `pd` = `put-down`, `st` = `stack`, `us` = `unstack`.

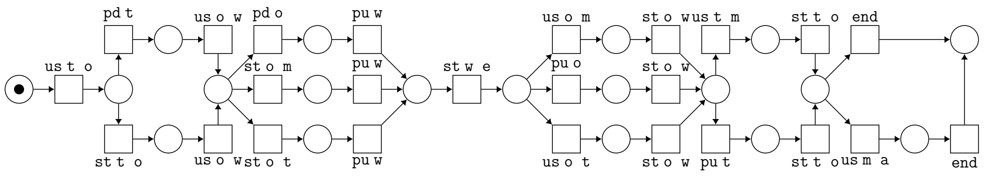


Figure 5: Petri net M_{G_1} learned from event log L_{G_1}.

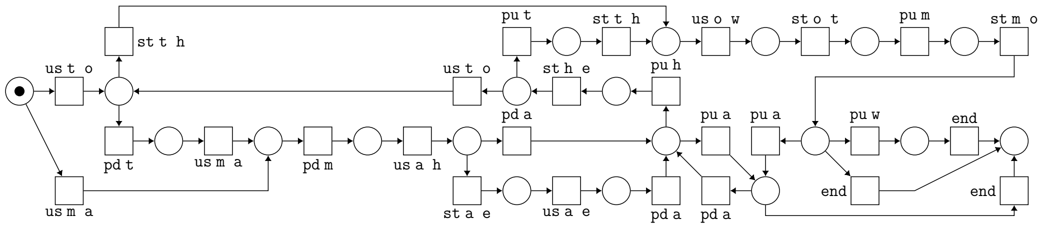


Figure 6: Petri net M_{G_2} learned from event log L_{G_2}.

the agent with each learned process model. Table 3 shows two example optimal alignments, denoted by γ_1 and γ_2, between a partially observed trace τ_o and models M_{G_1} and M_{G_2}. Trace $\tau_o = \langle$`(unstack t o)`, `(put-down t)`, `(unstack o w)`, `(stack o m)`, `(pick-up w)`, `(stack w e)`, `(unstack o m)`$\rangle$ consists of seven actions, denoted by $a_1, \ldots, a_7$, where action indices correspond to their positions in the trace, which do not provide a complete sequence of actions from the initial state to the goal state. Let the ground truth be that the agent that performed τ_o aims to achieve goal G_1. Model M_{G_1} describes a trace that can match τ_o perfectly. However, the best-matched trace described by M_{G_2} can only align three actions, a_1, a_3, and a_5 with τ_o, while a_2, a_4, a_6, and a_7 are asynchronous moves denoted by the special skip symbol "≫". The optimal alignments are computed using the procedure proposed by Adriansyah et al. [32], which aims to find a trace from process model M_{G_i} that requires the smallest number of asynchronous moves with the observed trace τ_o. The lower the number of asynchronous moves, the better the observed trace τ_o fits model M_{G_i}, which indicates the agent is more likely to achieve goal G_i. Intuitively, in the example in Table 3, the agent is more likely to work toward goal G_1. Next, the probability distribution over candidate goals G_1 to G_N is computed using optimal alignments γ_1 to γ_N that describes the likelihood of τ_o leading to the true goal.

		a_1	a_2	a_3	a_4	a_5	a_6	a_7
$\gamma_1 =$	τ_o	`(unstack t o)`	`(put-down t)`	`(unstack o w)`	`(stack o m)`	`(pick-up w)`	`(stack w e)`	`(unstack o m)`
	M_{G_1}	`(unstack t o)`	`(put-down t)`	`(unstack o w)`	`(stack o m)`	`(pick-up w)`	`(stack w e)`	`(unstack o m)`
i^δ		1^1	2^1	3^1	4^1	5^1	6^1	7^1
$c(\tau_o, M_{G_2}, i)$		0	0	0	0	0	0	0
$\gamma_2 =$	τ_o	`(unstack t o)`	`(put-down t)`	`(unstack o w)`	`(stack o m)`	`(pick-up w)`	`(stack w e)`	`(unstack o m)`
	M_{G_2}	`(unstack t o)`	`>>`	`(unstack o w)`	`>>`	`(pick-up w)`	`>>`	`>>`
i^δ		1^1	2^1	3^1	4^1	5^1	6^1	7^1
$c(\tau_o, M_{G_2}, i)$		0	1	0	1	0	1	1

Table 3: Optimal alignments between trace τ_o observed in environment 0 from Fig. 1a and models M_{G_1} and M_{G_2}.

Step 3: The PM-based GR system converts the optimal alignments between observed trace τ and the process models into a probability distribution indicating the likelihood of each candidate goal being pursued by the agent. Firstly, the *weight* of the constructed optimal alignment between τ and model M_G is computed as shown below.

$$\omega(\tau, M_G) = \phi + \lambda^m \times \sum_{i=1}^{n} \left(i^\delta \times c(\tau, M_G, i)\right), \text{ where} \tag{1}$$

- $c(\tau, M_G, i)$ is the cost of the *asynchronous move on trace* in position i in the alignment, which means that action a_i in the trace cannot be matched by a corresponding step in the model; see for instance actions a_2, a_4, a_6, a_7 in γ_2 in Table 3,
- i^δ, with $\delta \geq 0$, is a discount factor that emphasizes that the more recent disagreements between the trace and the model impact the alignment weight more,
- $\phi \geq 0$ is the "smoothening" constant that flattens the likelihoods of the goals in case of (close-to) perfect alignments for all (or most of) the models, and
- λ^m penalizes the suffix of the trace that deviates from the learned model, $\lambda \geq 1$ is a constant and m is the number of consecutive asynchronous moves on trace at the end of the optimal alignment; for example, a_6 and a_7 in γ_2 are two consecutive asynchronous moves on trace and, hence, it holds that $m = 2$.

Following the method by Masters and Sardiña [33], the computed alignment weights are used to compute the posterior probabilities of the agent pursuing each candidate goal. Given an observed trace τ, the probability of the agent pursuing goal G, from a set of candidate goals $\mathcal{G}$, that is $\Pr(G \mid \tau)$, is obtained as shown in Eq. (2). The PM-based GR system then infers the likely goal(s) according to the constructed probability distribution over all the goals. For more details on the PM-based GR system and its goal inference, please refer to the original work [21, 22].

$$\Pr(G \mid \tau) = \frac{e^{-\beta \times \omega(\tau, M_G)}}{\sum_{G' \in \mathcal{G}} e^{-\beta \times \omega(\tau, M_{G'})}}, \quad \text{where} \quad \beta = \frac{1}{1 + \min_{G \in \mathcal{G}} \omega(\tau, M_G)} \tag{2}$$

Note that the parameters ϕ, δ, and λ in Eq. (1) are configurable. In this paper, we use default parameters presented in [21] to configure the PM-based GR system, where $\phi = 50$, $\delta = 1.0$, $\lambda = 1.5$, and the cost of every asynchronous move on trace $c(\tau, M_G, i) = 1$. Thus, in the blocks-world example, the alignment weight $\omega(\tau_o, M_{G_1})$ is 50, since all the moves in γ_1 are synchronous and only the constant ϕ contributes to the weight. On the other hand, $\omega(\tau_o, M_{G_2}) = 50 + 1.5^2 \times (1 \times 2^1 + 1 \times 4^1 + 1 \times 6^1 + 1 \times 7^1) = 92.75$, since a_2, a_4, a_6, a_7 in γ_2 are asynchronous moves. Therefore, when incorporating the alignment weights $\omega(\tau_o, M_{G_1})$ and $\omega(\tau_o, M_{G_2})$ into Eq. (2), the probabilities of the agent achieving G_1 and G_2 are 0.69 and 0.31, respectively. The probabilities indicate that the observed trace τ_o is more likely to lead the agent to goal G_1, and, hence, the system infers G_1 as the goal of the agent.

4. Adaptive Process Mining-Based Goal Recognition in Changing Environments

The conventional PM-based GR system works in a static environment. However, if the environment changes from environment 0 to environment 1 in the running example, refer to Fig. 1, the agent must follow a new trace to achieve the same goal. Assume that the new observed sequence of actions for the agent acting in environment 1 to achieve G_1 is $\tau_i' = \langle$ `(put-down e)`, `(unstack m a)`, `(put-down m)`, `(unstack t o)`, `(stack t m)`, `(unstack a w)`, `(put-down a)` $\rangle$. If we align τ_i' to the models learned for the environment 0, M_{G_1} and M_{G_2}, we obtain optimal alignments γ_3 and γ_4, as shown in Table 4. The alignment weights, according to Eq. (1), are $\omega(\tau'_i, M_{G_1}) = 92.75$ and $\omega(\tau'_i, M_{G_2}) = 66$, respectively. According to Eq. (2), the inferred probabilities of the agent to achieve G_1 and G_2 are 0.40 and 0.60, respectively. Consequently, if the GR system relies on the models learned for environment 0, the newly inferred probabilities indicate that the agent executing τ'_i is more likely working toward G_2, which is incorrect given that the true goal of the agent is G_1. This running example illustrates the limitations of the conventional GR system when the underlying environment changes. Therefore, in this section, we extend the conventional PM-based GR system with control mechanisms governed by adaptive GR strategies that trigger the relearning of the process models to solve the adaptive GR problem.

		a_1	a_2	a_3	a_4	a_5	a_6	a_7
$\gamma_3 =$	τ'_i	`(put-down e)`	`(unstack m a)`	`(put-down m)`	`(unstack t o)`	`(stack t m)`	`(unstack a w)`	`(put-down a)`
	M_{G_1}	`>>`	`>>`	`>>`	`(unstack t o)`	`(stack t m)`	`>>`	`>>`
i^δ		1^1	2^1	3^1	4^1	5^1	6^1	7^1
$c(\tau'_i, M_{G_1}, i)$		0	0	0	0	0	0	0
$\gamma_4 =$	τ'_i	`(put-down e)`	`(unstack m a)`	`(put-down m)`	`(unstack t o)`	`(stack t m)`	`(unstack a w)`	`(put-down a)`
	M_{G_2}	`>>`	`(unstack m a)`	`(put-down m)`	`>>`	`>>`	`>>`	`(put-down a)`
i^δ		1^1	2^1	3^1	4^1	5^1	6^1	7^1
$c(\tau'_i, M_{G_2}, i)$		1	0	0	1	1	1	0

Table 4: Optimal alignments between trace prefix τ'_i observed in environment 1 from Fig. 1a and models M_{G_1} and M_{G_2}.

The architecture of the adaptive version of the PM-based GR system consists of a conventional PM-based GR system and two additional components, *feedback collection* and *relearn decision*, as shown in Fig. 7. The adaptive GR system receives observed trace prefixes as input. Each time the system receives a newly observed sequence of actions, it infers the likely goal(s) by following the principles described in Section 3.

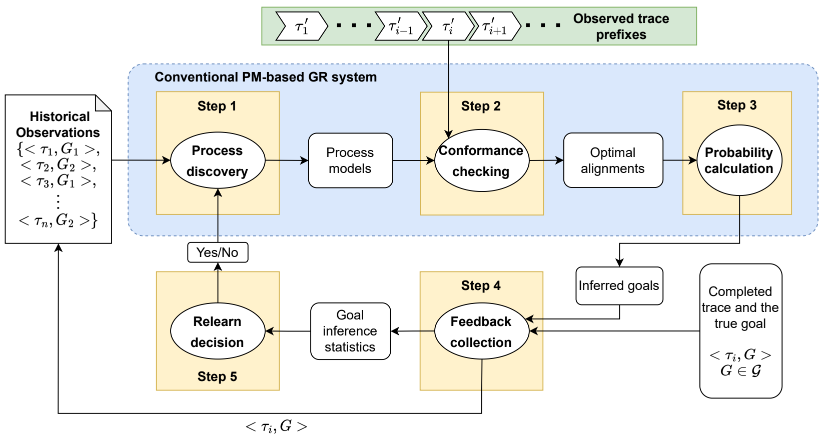


Figure 7: Architecture of the adaptive PM-based GR system.

Next, we discuss the two new steps of the the adaptive version of the PM-based GR system.

Step 4: Once the agent reaches a goal, the information on completed trace τ_i, true goal G the agent has achieved by executing τ_i, and the previously inferred goal based on the corresponding trace prefix τ'_i are passed to the feedback collection component. We assume that once the agent reaches a goal, that is, completes a trace of actions, the corresponding true goal can be detected. The feedback collection component performs two tasks. First, it evaluates the inferred goal against the true goal. The evaluation results are then communicated to the relearn decision component through goal inference statistics to inform whether to retrain the GR system. Second, the completed trace and the corresponding achieved goal are stored in the collection of historical observations, ready to be used to relearn the process models once the necessity arises.

Step 5: The relearn decision component uses the goal inference statistics to generate signals to instruct the GR system to update the discovered process models, that is, to implement adaptive GR strategies. In this work, we explore open- and closed-loop strategies. The open-loop strategy instructs the process discovery component to relearn the models after a specified number of performed GR tasks; the decision to relearn the models is triggered regularly. On the other hand, a closed-loop strategy evaluates the historical performance of GR inferences by the system summarized in the statistics generated by the feedback collection component to decide when to trigger the decision to relearn the process models. We use balanced accuracy to measure the GR performance, as explained in Section 5.2. Intuitively, a drop in the GR accuracy indicates that the environment may have changed. Consequently, the adaptive GR system may decide to relearn the process models from the recently observed historical traces.

In this work, we present and evaluate three adaptive GR strategies summarized below.

- **Open-loop adaptive GR strategy.** This strategy requests automatically relearning the process models from the most recent historical observations after a pre-configured number of GR inferences, regardless of the GR performance.
- **Closed-loop adaptive GR strategy based on average accuracy.** This strategy monitors the average accuracy over the pre-configured number of most recent GR inferences. If the average accuracy drops below a given threshold, it requests to relearn the process models from the most recent historical observations.
- **Closed-loop adaptive GR strategy based on accuracy trend.** This strategy uses linear regression to analyze the accuracy trend of the most recent GR inferences. If this trend suggests that the accuracy of a number of the subsequent GR inferences will drop below a given threshold, the strategy requests to relearn the process models from the most recent historical observations.

5. Evaluation

This section presents an evaluation we conducted to study the performance of our adaptive GR systems. Section 5.1 describes the synthetic datasets we generated using the extended version of the GRACE tool and the real-world event logs we used to evaluate the GR performance. Section 5.2 presents the measures we used to assess the performance of our GR systems. Finally, Section 5.3 describes the experiments, while Section 5.4 discusses the results of the experiments.

5.1. Datasets

The synthetic dataset, a collection of adaptive GR problems, used in our experiments was generated using the extended version of the GRACE tool [25]. The tool takes a static GR problem instance described in the Planning Domain Definition Language (PDDL) [34] and a configuration as input and generates a sequence of GR problems, each composed of a PDDL description of the environment and an observation of an agent accomplishing a goal in the environment, as output. The configuration specifies changes that should be applied in the environment of the input GR problem (e.g., a change of the initial state of the agent and objects in the environment) and a drift type according to which this input environment should evolve in the modified environment.

Each problem in the generated sequence is either the original GR problem or the GR problem over the changed environment. Two consecutive problems in the sequence specify their temporal relation; the preceding problem should be solved before the succeeding problem. The order of the problems in the sequence implements the requested drift type.

To ensure that the GRACE tool generates environments significantly different from those in the input GR problems, we extended the tool to use the significance of change (SOC) measure. The SOC measure quantifies the differences between the original and modified environments. In the first implementation of the GRACE tool [25], the input environment is modified randomly. For example, a new initial state of the agent is discovered via a random walk from the original initial state of the environment. The new initial state discovered this way could be similar to the original initial state. To avoid such situations, we extended GRACE to use the differences between optimal plans in the original and the changed environments to quantify how significantly the environment has been modified. The difference between two given plans is computed as their *Levenshtein edit distance* [35]. Let the original environment have n goal candidates, $G_1, \ldots, G_n$, and an initial state I. For each goal candidate G_i, $i \in [1 .. n]$, we find an optimal plan π_i that starts in I to G_i using the top-k planner [36]. The GRACE tool modifies the original environment to an environment with n candidate goals, namely $G'_1, \ldots, G'_n$, and one initial state I'. We compute optimal plans $\pi'_1, \ldots, \pi'_n$ that start in I' to goals $G'_1, \ldots, G'_n$, where π'_i is an optimal plan from I' to G'_i. By $dist(\pi_i, \pi'_i)$, we denote the Levenshtein edit distances between plans π_i and π'_i; note that the plans are captured as sequences of actions performed by the agent, that is, traces. If two traces, π_i and π'_i, are entirely different, the maximum possible Levenshtein edit distance between them can be obtained by summing their lengths, that is $|\pi_i| + |\pi'_i|$. Equation (3) defines the SOC measure between the original and modified environments.

$$SOC = \frac{\sum_{i=1}^{n} dist(\pi_i, \pi'_i)}{\sum_{i=1}^{n} \left(|\pi_i| + |\pi'_i|\right)} \tag{3}$$

It holds that SOC is between zero and one, inclusively, that is, $SOC \in [0, 1]$. The larger the SOC value, the more significant the difference between the original and modified environments.

We used conventional GR problem instances from 15 International Planning Competition (IPC) domains [1] as input to GRACE. For each input problem instance, GRACE was configured to generate the corresponding adaptive GR problem instance by following these four steps:

1. Modify the initial state of the environment of the input GR problem to generate a new GR environment that is significantly different from the input environment (the SOC value greater or equal to 0.25).
2. Use the top-k planner [36] to construct 1 000 plans from the initial state to every goal candidate state, both for the original and modified environment.
3. Use the original and modified environments to simulate sudden, gradual, and reoccurring drifts from the original to the modified environment as sequences of these two environments.
4. For each environment at every position in the sequence, select one generated plan towards each candidate goal from the set of constructed 1 000 plans to represent the behavior of the agent for accomplishing the goal in that environment at that time step.

We constructed and made publicly available 228 adaptive GR problem instances from 15 domains (76 problem instances times 3 types of drifts).[3] Note that the number of problem

[3] https://doi.org/10.26188/21802081

instances is determined by the number of different problems in the original IPC dataset. Figure 1 shows the original environment (environment 0) and the modified environment (environment 1) of one constructed adaptive GR problem instance from the constructed datasets, while Fig. 2 shows how these two environments are arranged in a sequence by GRACE to simulate a gradual drift from environment 0 to environment 1.

To conduct experiments over real-world datasets, we used the preprocessed event logs provided by Teinemaa et al. [37], in which the traces are grouped by Linear Temporal Logic (LTL) classifiers. The traces within each group accomplish the same goal. For each event log, we sorted the traces based on the timestamps of their last events. Following this sorted chronological order, we selected an equal number of traces that accomplish each goal candidate to simulate a sequence of observed traces as inputs for GR tasks. We then extracted traces from the very beginning and the very end of each log to induce a drift. Intuitively, a time gap between these two trace groups may result in a noticeable concept drift. We then used the first ten time steps (where each time step is represented by the agent traces towards all the goal candidates) for discovering the initial process models. We used the subsequent 50 time steps (time steps 11 to 60) to simulate GR tasks within the initial environment and the last 50 time steps to simulate GR tasks after the changes in the initial environment. Finally, we conducted experiments using the conventional GR system that does not adapt to environmental changes over the constructed 100 testing time steps. If the average recognition accuracy over the first 50 time steps was significantly higher (by 10%) than the average accuracy over the last 50 time steps, we asserted that the concept drift indeed occurred. Using this approach, we selected two event logs from the real-world business domains that exhibit concept drift: BPIC 2011 [4] and Hospital Billing [5]. Note that for the Sepsis Cases event log [6], the number of traces in the event log is relatively small, resulting in 98 time steps after preprocessing. Thus, we examined the first and the last 20 time steps to ascertain whether there was a significant drop in GR accuracy over these two periods. The results revealed a 0.061 (8.6%) accuracy drop, but we included the Sepsis Cases event log in the subsequent experiments. Consequently, we obtained three event logs[3] from real-world business domains, specifically BPIC 2011, Hospital Billing, and Sepsis Cases, which we used for evaluating our adaptive GR systems.

5.2. *Performance Measures*

For each observation of an agent attempting to accomplish a goal supplied with the adaptive GR problem instance from the dataset, the GR performance is measured using the *balanced accuracy* measure [38], which requires four terms to compute. True Positive (*TP*) is the number of correct goals inferred by the GR system. True Negative (*TN*) is the number of incorrect goals not inferred. False Positive (*FP*) is the number of incorrect goals inferred by the GR system. Finally, False Negative (*FN*) is the number of correct goals not inferred. For example, consider a GR system observing an agent that executes a sequence of actions, working towards a true hidden goal (G_1) among ten candidate goals (G_1 to G_{10}). Suppose that, according to the observed action sequence, the GR system infers G_1 and G_2 as possible goals the agent is trying to achieve. Therefore, G_1 and G_2 are two positive goals, and the rest of the candidate goals are negative goals. In this scenario, for the two positive goals, G_1 is the true hidden goal correctly inferred

[4]https://doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54
[5]https://doi.org/10.4121/uuid:76c46b83-c930-4798-a1c9-4be94dfeb741
[6]https://doi.org/10.4121/uuid:915d2bfb-7e84-49ad-a286-dc35f063a460

by the GR system. Thus, TP is equal to one. Goal G_2 is not the true hidden goal; it is falsely inferred by the GR system. Thus, FP equals one. None of the eight negative goals (G_3 to G_{10}) is the true hidden goal. Hence, TN equals eight because the GR system made the correct decision not to infer these goals. Finally, FN is equal to zero because none of the true hidden goals are missed; the GR system correctly recognized the true hidden goal. Note that, in our experiments, $TP, FN \in \{0, 1\}$, as there is only one true goal per instance, while $TN, FP \in \{0, \dots, |\mathcal{G}| - 1\}$, where $\mathcal{G}$ stands for the set of candidate goals. Given the four terms (TP, TN, FP, and FN), the balanced accuracy ($BACC$) is an average of the true positive and the true negative rates, given below.

$$BACC = \frac{1}{2} \times \left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right) \tag{4}$$

5.3. Experiments

We evaluated the performance of the standard PM-based GR system (see Section 3) and the three adaptive GR strategies (see Section 4) over the synthetic and real-world instances of the adaptive GR problems. We denote the evaluated GR systems that follow different adaptive strategies as following: SYS_{std} is the standard PM-based GR system; SYS_{ol} is the adaptive PM-based GR system that implements the open-loop adaptive GR strategy; SYS_{cla} is the adaptive PM-based GR system that implements the closed-loop adaptive GR strategy based on average accuracy; and SYS_{clt} is the adaptive PM-based GR system that implements the closed-loop adaptive GR strategy based on accuracy trend.

We used the performance of SYS_{std} as the baseline to compare it with the performance of the three adaptive GR systems. In SYS_{ol}, we set the system to relearn after every ten GR inferences. In SYS_{cla}, we set the relearn threshold of the highest average accuracy over ten consecutive GR inferences, denoted by ACC_{h10}, to 80%. Hence, the system would relearn if the average accuracy over the past ten GR inferences is below $0.8 \times ACC_{h10}$. In SYS_{clt}, if the predicted accuracy of the following ten GR inferences is below $0.8 \times ACC_{h10}$, then the GR system relearns the process models. We simulated partial observability of the agent in each GR task by only including 50% of the actions from each trace towards a goal supplied in the datasets.

All the adaptive GR problem instances from the datasets were solved using our open-source implementation of the PM-based GR system and the proposed adaptive GR strategies on a single core of an Intel Xeon Processor (Skylake, IBRS) @ 2.0GHz with 16GB of RAM.[7]

5.4. Results

The results of the experiments confirmed that, compared to the standard PM-based GR system, the adaptive GR systems achieve better recognition accuracies. Figure 8 summarizes the performance of the four evaluated GR systems (SYS_{std}, SYS_{ol}, SYS_{cla}, and SYS_{clt}) on an adaptive GR problem instance from the synthetic zeno-travel domain with a sudden drift from the original to the modified environment. Specifically, the first 50 GR problems are defined for the original environment, while problems 51 to 100 are defined for the modified environment. In each plot in Fig. 8, the X-axis shows a sequence of problems tackled by the GR system, while the Y-axis shows the average balanced accuracy of the recognized goals. Note that each adaptive GR problem instance in the dataset is defined for multiple (n) candidate goals (G_1 to

[7]The code to run the experiments and our implementations of the GR systems can be accessed here: https://doi.org/10.26188/25329490.

G_n). Recall that for each GR problem at position j in the sequence on the X-axis, we have n observations (i.e., traces of actions), one trace the agent has followed for each goal. By $BACC^s_{G_i}$, we denote the balanced accuracy of performing goal recognition using GR system s on the trace towards goal G_i, $i \in [1\,..\,n]$, $s \in \{SYS_{std}, SYS_{ol}, SYS_{cla}, SYS_{clt}\}$. Hence, each data point in the plots stands for $\left(\sum_{i=1}^{n} BACC^s_{G_i}\right)/n$ and is denoted by $ABACC^s_j$. The line segment associated with each data point represents the corresponding standard deviation range, while data points in red, in addition, represent that at that position in the sequence, the system decided to update its knowledge base and relearned the models it uses for solving GR problems.

Figure 8a shows that the average recognition accuracy drops from 0.927 to 0.670 after the environment changes if the GR system does not update the process models. In contrast, for the GR systems with relearn mechanisms, the recognition accuracy can recover to some degree after the environment changes. The open-loop adaptive GR system SYS_{ol} (Fig. 8b) relearns ten times, and the average accuracy after environment change is 0.789. The closed-loop adaptive GR system based on average accuracy SYS_{cla} (Fig. 8c) relearns once, and the average accuracy after environment change is 0.761. Finally, the closed-loop adaptive GR system based on accuracy trend SYS_{clt} (Fig. 8d) relearns four times, and the average accuracy after environment change is 0.792. The detailed results for each adaptive GR system, performing in different problem instances under three types of environmental drift, are available online.[8]

We computed the average accuracies of the goal recognition for the four evaluated GR systems in the original and modified environments over all the problem instances and all the drift types in the datasets. The average accuracy achieved by GR system s in the original environment is denoted by $ABACC0^s$ and equals $\left(\sum_{j \in E0} ABACC^s_j\right)/|E0|$, while $ABACC1^s$ stands for the average accuracy in the modified environment and equals $\left(\sum_{j \in E1} ABACC^s_j\right)/|E1|$; $E0$ and $E1$ are the sets of all positions in the sequence of GR problems defined over the original environment and the modified environment, respectively. The difference between the average accuracies in the original and modified environments for the SYS_{std} system shown in Fig. 8a (i.e., the system without relearn capabilities) is the drop in accuracy for the baseline system, see below.

$$ACCDROP = ABACC0^{SYS_{std}} - ABACC1^{SYS_{std}} \tag{5}$$

The accuracy improvement is the difference between the average accuracies in the modified environments computed for adaptive GR system s and the standard GR system SYS_{std}.

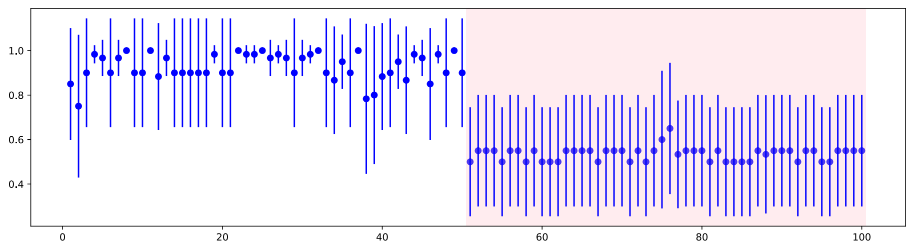
$$Improvement^s = ABACC1^s - ABACC1^{SYS_{std}}, \quad s \in \{SYS_{ol}, SYS_{cla}, SYS_{clt}\} \tag{6}$$

Next, the improvement ratio for GR system s is defined as follows.

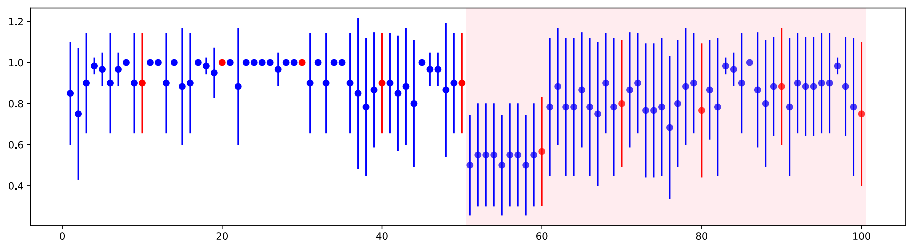
$$ImprovementRatio^s = \frac{Improvement^s}{ACCDROP}, \quad s \in \{SYS_{ol}, SYS_{cla}, SYS_{clt}\} \tag{7}$$

Table 5 shows accuracy improvement ratios computed per drift type for the three adaptive GR systems and the average number of times (per problem instance) the system triggered relearning of the knowledge base. The detailed results of all the experiments based on which the data in the table was computed are publicly available.[8] For instance, the table suggests that the open-loop system on a problem with a sudden drift relearns (on average) ten times and demonstrates the improvement ratio of 0.62. That is, it recovers (on average) 62% of the drop in recognition

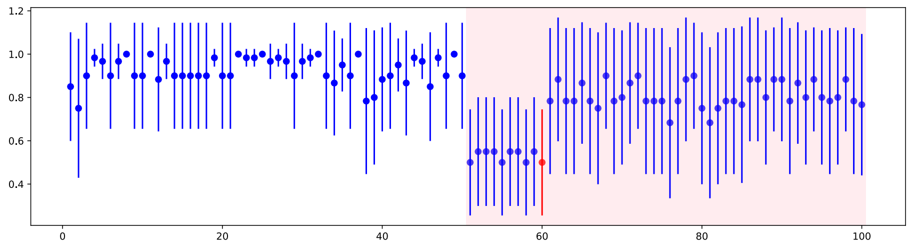
[8] https://doi.org/10.26188/21901296



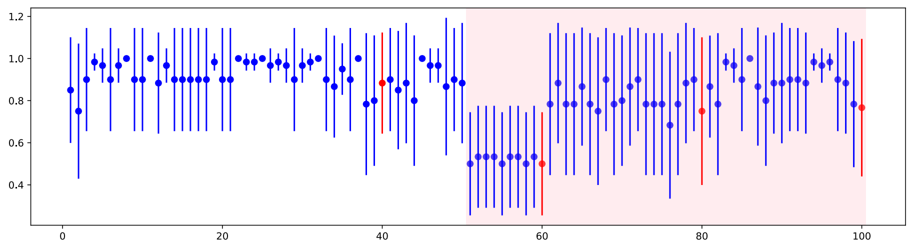
(a) The standard PM-based GR system SYS_{std}; average accuracy in cases 1–50 and 51–100 is 0.927 and 0.670, respectively, with no updates of the process models.



(b) The open-loop adaptive PM-based GR system SYS_{ol}; average accuracy in cases 1–50 and 51–100 is 0.943 and 0.789, respectively, with ten updates of the process models.



(c) The closed-loop adaptive PM-based GR system based on average accuracy SYS_{cla}; average accuracy in cases 1–50 and 51–100 is 0.927 and 0.761, respectively, with one update of the process models.



(d) The closed-loop adaptive PM-based GR system based on accuracy trend SYS_{clt}; average accuracy in cases 1–50 and 51–100 is 0.921 and 0.792, respectively, with four updates of the process models.

Figure 8: Performance of four GR systems on a single adaptive GR problem instance with a sudden drift.

accuracy due to the change of the environment. The last column in Table 5 shows the absolute accuracy drop for each type of drift over all problem instances. We conclude from the results that the proposed adaptive GR systems can improve recognition accuracy compared to the standard PM-based GR system. In general, the more times the system relearned, the better improvement in accuracy was observed. However, good improvements in accuracy can already be obtained by retraining the system a small number of times. Hence, it is evident that a closed-loop GR system can achieve good performance over a wide range of domains and drift types while keeping the number of relearn episodes low.

Drift	Open-loop	Closed-loop (Avg)	Closed-loop (Trend)	*ACCDROP*
Sudden	62.0% / 10	33.5% / 1.4	51.6% / 2.3	0.170
Gradual	106.2% / 15	43.6% / 1.7	87.6% / 3.2	0.170
Reoccuring	62.3% / 20	36.1% / 3.4	51.5% / 5.4	0.167

Table 5: Accuracy improvement ratio and the average number of times the process models were relearned (improvement ratio/avg. number of relearn episodes per problem instance) for three adaptive GR systems over 218 problem instances.

The improvement ratios in GR accuracy achieved by using adaptive GR systems, along with the number of times the system triggered relearning of process models for the real-world business domains—BPIC 2011, Sepsis Cases, and Hospital Billing—are displayed in Table 6. The column labeled "*ACCDROP*" represents the absolute accuracy drop after the environmental change (concept drift) when using SYS_{std}. Nevertheless, these performance drops can be mitigated using the adaptive GR systems: SYS_{ol}, SYS_{cla}, and SYS_{clt}. For instance, for the BPIC 2011 event log, the average accuracy drops by 0.116 when using SYS_{std} due to the environmental change, and SYS_{ol} can recover 122.4% of the accuracy drop by relearning the process models ten times over the entire goal recognition period.

Domain	Open-loop	Closed-loop (Avg)	Closed-loop (Trend)	*ACCDROP*
BPIC 2011	122.4% / 10	95.7% / 6	152.6% / 7	0.116 (from 0.540 to 0.424)
Sepsis Cases	98.4% / 10	173.8% / 2	123.0% / 1	0.061 (from 0.713 to 0.652)
Hospital Billing	108.3% / 10	96.4% / 4	137.0% / 3	0.084 (from 0.720 to 0.636)

Table 6: Accuracy improvement ratio and the average number of times the process models were relearned (improvement ratio/avg. number of relearn episodes per problem instance) for three adaptive GR systems over three real-world domains.

6. Related Work

Adaptive GR demands GR systems to tackle multiple "single-shot" GR tasks over a period of time. When the environment changes, the GR systems need to detect the changes and subsequently adapt their GR inference procedures. The adaptive GR problem can be decomposed into subproblems of three types: a conventional GR problem, a knowledge model updating problem, and a concept drift detection problem. In this section, we discuss related works on these subproblems, describe promising areas for applying the adaptive GR techniques, and mention useful third-party tools used in this work.

Existing conventional GR techniques can be categorized into three main categories: plan library-based GR [5, 17, 39], planning-based GR [26, 40, 41], and learning-based GR [21, 22, 42] approaches. The plan library-based GR approaches rely on a set of standard plans to accomplish goal candidates. The plan library can be formulated in different formats, such as action taxonomy [5] or markov logic networks [39]. The plan library-based approaches require human effort to handcraft models, and these models struggle to generalize to unseen traces effectively. Planning-based GR approaches involve using planners to identify an optimal plan and subsequently comparing this optimal plan to the observed action sequence to infer possible goals. However, these approaches require domain experts to create the domain models manually. The learning-based GR approaches can autonomously acquire knowledge models, such as LSTM [42] or Petri nets [21, 22], from historical observations. These approaches do not rely on domain experts, and the learned models have the capacity to generalize to yet unseen traces.

Lesh [43] proposed an approach for selecting the best-performing adapted GR system for solving a single-shot GR problem, given a collection of allowed GR system adaptations. In contrast, our work studies the problem of retraining a data-driven GR system to sustain its high GR performance over an extended period. The work by Bryce et al. [44] proposed a technique for issuing queries for updating the knowledge models. This technique adapts the models by analyzing query answers. However, instead of updating the models automatically, this technique relies on human-provided query answers. Chakraborti et al. [45] proposed a model reconciliation algorithm. The idea of model reconciliation is that if two different models result in different optimal plans for the same goal, the reconciliation technique attempts to modify one of the models to get a single optimal plan for both models. Consequently, when reconciling the models, one updates them according to a single trace. In contrast, our relearning mechanism updates the models based on an arbitrary number of traces. The algorithms for process model repair can be used for updating the process models of the PM-based GR systems [46, 47]. In this work, however, the PM-based GR systems relearn new process models from scratch. We use Directly Follows Miner [31] to learn the initial models and relearn the up-to-date models.

The problem of concept drift detection in process mining studies ways to detect behavioral changes by analyzing the chronologically ordered traces in an event log [24, 48, 49]. The solutions to the concept drift detection problem can be used to trigger the relearning of knowledge models of adaptive GR systems. In this work, we used feedback on GR performance to detect changes in the environment. Future work will study the usefulness of concept drift detection techniques from the area of process mining for solving adaptive GR problems.

The adaptive GR systems can be deployed in scenarios such as human-robot teaming, where robots continuously update their estimation of the plans and goals of the human(s) in the loop with every new observation. The domain knowledge of an adaptive GR system can evolve over time due to changing human goals and a changing environment. An early example using the planning-based GR system [19] and an abstraction technique called "resource profiles" is presented by Chakraborti et al. [50], where the evolving nature of the environment was extracted from the output of the GR algorithm to feed into the ultimate plan generator. Similar approaches have recently been built on this principle of abstraction in settings that require higher-order representations driving the adaptive plan/goal recognition and plan generation cycle [51].

The experimental data for evaluating adaptive GR systems is a series of historical agents' traces in which the traces were generated in different environments. Such data can be generated by the GRACE tool [25]. Given a GR problem instance as input, captured in PDDL [34], GRACE uses the Tarski [52] and LAPKT [53] tools to parse and manipulate the input environment models to simulate different drifts and the top-k [36] and diverse [54] planners to generate sequences of agent actions towards different goals in the drifting environments.

7. Discussion and Limitations

PM-based GR systems learn process models based on historical observations and use these models to infer intelligent agents' goals. As such systems are data-driven, they suit well for real-world scenarios [22], in contrast to planning- and plan library-based systems that rely on human-crafted inputs. The GR performance in real-world business domains, as presented in Section 5.4, further substantiates the applicability of the presented adaptive PM-based GR systems in real-world settings. For example, in Sepsis Cases, our approach can aid in forecasting whether a patient may require intensive care, providing crucial information to assist hospitals in delivering more effective and safer treatment. Another promising real-world application of the techniques presented in this paper is the implementation of a system that supports a disabled person using a robotic prosthesis to perform various movements [55]. In this project, we applied the PM-based GR techniques to recognize the movement goals of the person by analyzing the signals they generate. Such a system can benefit from the ability to detect changes in the behavior of the person and adapt its GR practices to ensure proper control of the robotic prosthesis. Compared to the learning-based GR based on LSTM networks [42], the PM-based systems can be expected to require less training data to perform well [22], so the relearning can also be done fast.

A process discovery algorithm aims to construct a process model that generalizes well to future traces, would they be generated, that are not contained in the event log the model was discovered from [27]. The PM-based GR system capitalizes on this ability of discovered models to describe yet unseen traces. Specifically, if an agent follows a never observed trace toward a goal, assuming discovered process models generalize well, the model learned from historical traces to that goal should describe the trace. Consequently, this trace will align well with the model, suggesting it is the true goal the agent currently pursues.

We acknowledge a range of limitations of our work so far. Existing GR techniques require the knowledge of goal candidates and only select the inferred goals from these known candidates. However, in practice, intelligent agents may aim to achieve fresh goals that they never achieved in the past. Identification of new goals of observed agents is an interesting direction for future work. We assume that the feedback on the solution to each GR problem can be obtained (in the form of the true goal the agent eventually achieved) before the next GR problem is to be solved. In future work, one can generalize our setup to account for situations when certain true goals cannot be attained, or the next GR problem must be solved before the solutions or true goals for the previous problems are available. Moreover, the three adaptive GR systems presented in Section 4 can be further refined. For example, when the systems update the learned process models, they relearn new models from scratch. An alternative, arguably more efficient, way forward is to use process model repair techniques [46, 47] to update the models incrementally. Finally, the performance of the presented adaptive GR systems relies on configurable parameters, such as ϕ, δ, λ (refer to Section 3), and the threshold for deciding whether to relearn the process models (refer to Section 4). In this work, we used the default parameters [21, 22], which demonstrated to be sufficient to support the conclusion that the adaptive GR systems can improve the overall GR performance compared to the conventional GR systems. However, exploring the optimal parameter configurations can yield higher overall GR performance. An interesting direction for future work would be to investigate parameter optimization methods that aim to improve the overall GR performance while minimizing the costs of relearning the process models.

8. Conclusion

In this article, we studied the goal recognition (GR) problem over extended periods in which the environment in which the observed agent operates may change. We first defined the *adaptive* GR problem in an abstract manner that suits, among others, standard definitions of probabilistic goal recognition. Roughly speaking, the *adaptive* GR problem amounts to solving the current GR instance task *in the context of previously solved problem instances*; the various instances may (slightly) differ in the underlying environment as well in the true agent's goal. We then presented three strategies for adaptive GR systems to address this challenging problem. To accurately recognize the goals in changing environments, the proposed GR systems were designed as both open- and closed-loop control systems that measure their recognition accuracy over time and react by relearning their knowledge base if the recognition accuracy becomes unsatisfactory. We conducted an experimental evaluation to validate the effectiveness of our proposed GR systems in addressing *adaptive* GR problems and showed that, indeed, they achieve better overall recognition accuracies across the target time intervals. Furthermore, we compared our (adaptive) GR systems based on the *effort* they require to achieve their performance, as the number of times they update their knowledge base. In this work, the GR system is grounded in process mining techniques [21, 22]. The experiments were done over both synthetic and real-world domains. To do so, we presented a tool (GRACE) for generating adaptive GR problem instances with significant changes—drifts—in the environment, and generated a dataset that we made publicly available as a benchmark to support future research on adaptive goal recognition.

References

[1] D. Dennett, The Intentional Stance, MIT Press, 1987.

[2] M. E. Bratman, Intentions, Plans, and Practical Reason, Harvard University Press, 1987.

[3] C. F. Schmidt, N. S. Sridharan, J. L. Goodson, The plan recognition problem: An intersection of psychology and artificial intelligence, Artificial Intelligence 11 (1978) 45–83. doi:https://doi.org/10.1016/0004-3702(78)90012-7.

[4] G. Sukthankar, C. Geib, H. H. Bui, D. Pynadath, R. P. Goldman, Plan, activity, and intent recognition: Theory and practice, Newnes, 2014.

[5] H. A. Kautz, J. F. Allen, Generalized plan recognition, in: AAAI, Morgan Kaufmann, 1986.

[6] G. Sukthankar, K. Sycara, A cost minimization approach to human behavior recognition, 2005, pp. 1067–1074.

[7] A. Kott, W. M. McEneaney, Adversarial reasoning: Computational approaches to reading the opponent's mind, CRC Press, 2006.

[8] S. Carberry, Plan Recognition in Natural Language Dialogue, MIT Press, Cambridge, MA, USA, 1990.

[9] T. Holtgraves, Automatic intention recognition in conversation processing, Journal of Memory and Language 58 (2008) 627–645. URL: https://www.sciencedirect.com/science/article/pii/S0749596X07000721. doi:https://doi.org/10.1016/j.jml.2007.06.001.

[10] J. Tian, Z. Tu, N. Li, T. Su, X. Xu, Z. Wang, Intention model based multi-round dialogue strategies for conversational AI bots, Applied Intelligence 52 (2022) 13916–13940. URL: https://link.springer.com/10.1007/s10489-022-03288-8. doi:10.1007/s10489-022-03288-8.

[11] P. C. Roy, A. Bouzouane, S. Giroux, B. Bouchard, Possibilistic activity recognition in smart homes for cognitively impaired people, Applied Artificial Intelligence 25 (2011) 883–926.

[12] E. Demeester, M. Nuttin, D. Vanhooydonck, H. Van Brussel, A model-based, probabilistic framework for plan recognition in shared wheelchair control: Experiments and evaluation, in: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, volume 2, 2003, pp. 1456–1461.

[13] S. Lefèvre, D. Vasquez, C. Laugier, A survey on motion prediction and risk assessment for intelligent vehicles, Robomech Journal 1 (2014) 1–14.

[14] J. Firl, Q. Tran, Probabilistic maneuver prediction in traffic scenarios, in: ECMR, 2011, pp. 89–94.

[15] J. Kooij, N. Schneider, F. Flohr, D. Gavrila, Context-based pedestrian path prediction, 2014, pp. 618–633.

[16] N. Lesh, C. Rich, C. L. Sidner, Using plan recognition in human-computer collaboration, 1999, pp. 23–32.

[17] E. Charniak, R. P. Goldman, A bayesian model of plan recognition, Artif. Intell. 64 (1993).

[18] G. Sukthankar, K. P. Sycara, A cost minimization approach to human behavior recognition, in: AAMAS, ACM, 2005.
[19] M. Ramírez, H. Geffner, Probabilistic plan recognition using off-the-shelf classical planners, in: AAAI, AAAI Press, 2010.
[20] R. F. Pereira, N. Oren, F. Meneguzzi, Landmark-based approaches for goal recognition as planning, Artif. Intell. 279 (2020).
[21] A. Polyvyanyy, Z. Su, N. Lipovetzky, S. Sardiña, Goal recognition using off-the-shelf process mining techniques, in: AAMAS, IFAAMAS, 2020.
[22] Z. Su, A. Polyvyanyy, N. Lipovetzky, S. Sardiña, N. van Beest, Fast and accurate data-driven goal recognition using process mining techniques, Artificial Intelligence (2023) 103973. doi:https://doi.org/10.1016/j.artint.2023.103973.
[23] D. Singh, S. Sardiña, L. Padgham, G. James, Integrating learning into a BDI agent for environments with changing dynamics, in: C. K. Toby Walsh, C. Sierra (Eds.), IJCAI, 2011, pp. 2525–2530.
[24] A. Yeshchenko, C. D. Ciccio, J. Mendling, A. Polyvyanyy, Visual drift detection for event sequence data of business processes, IEEE Trans. Vis. Comput. Graph. 28 (2022).
[25] Z. Su, A. Polyvyanyy, N. Lipovetzky, S. Sardiña, N. van Beest, GRACE: A simulator for continuous goal recognition over changing environments, in: PMAI@IJCAI, volume 3310 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 37–48.
[26] M. Ramirez, H. Geffner, Probabilistic plan recognition using off-the-shelf classical planners, 2010, pp. 1121–1126.
[27] W. M. P. van der Aalst, Process Mining—Data Science in Action, Second Edition, Springer Berlin Heidelberg, 2016. doi:10.1007/978-3-662-49851-4.
[28] W. M. P. van der Aalst, T. Weijters, L. Maruster, Workflow mining: Discovering process models from event logs, IEEE Trans. Knowl. Data Eng. 16 (2004) 1128–1142.
[29] W. M. P. van der Aalst, The application of petri nets to workflow management, J. Circuits Syst. Comput. 8 (1998) 21–66.
[30] A. Polyvyanyy, A. Moffat, L. García-Bañuelos, Bootstrapping generalization of process models discovered from event data, in: CAiSE, volume 13295 of *LNCS*, Springer, 2022, pp. 36–54.
[31] S. J. Leemans, E. Poppe, M. T. Wynn, Directly follows-based process mining: Exploration & a case study, in: 2019 International Conference on Process Mining (ICPM), IEEE, 2019, pp. 25–32.
[32] A. Adriansyah, N. Sidorova, B. F. van Dongen, Cost-based fitness in conformance checking, in: ACSD, IEEE Computer Society, 2011, pp. 57–66.
[33] P. Masters, S. Sardiña, Cost-based goal recognition for path-planning, in: AAMAS, ACM, 2017, pp. 750–758.
[34] P. Haslum, N. Lipovetzky, D. Magazzeni, C. Muise, An Introduction to the Planning Domain Definition Language, Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2019.
[35] V. I. Levenshtein, et al., Binary codes capable of correcting deletions, insertions, and reversals, in: Soviet physics doklady, volume 10, Soviet Union, 1966, pp. 707–710.
[36] D. Speck, R. Mattmüller, B. Nebel, Symbolic top-k planning, in: AAAI, AAAI Press, 2020.
[37] I. Teinemaa, M. Dumas, M. L. Rosa, F. M. Maggi, Outcome-oriented predictive process monitoring: Review and benchmark, ACM Trans. Knowl. Discov. Data 13 (2019) 17:1–17:57.
[38] J. P. Mower, PREP-mt: predictive RNA editor for plant mitochondrial genes, BMC Bioinformatics 6 (2005). doi:10.1186/1471-2105-6-96.
[39] E. Ha, J. P. Rowe, B. W. Mott, J. C. Lester, Goal recognition with markov logic networks for player-adaptive games, in: AAAI, AAAI Press, 2012.
[40] R. F. Pereira, N. Oren, F. Meneguzzi, Landmark-based approaches for goal recognition as planning, Artificial Intelligence 279 (2020) 103217.
[41] L. R. A. Santos, F. Meneguzzi, R. F. Pereira, A. G. Pereira, An lp-based approach for goal recognition as planning, in: AAAI, AAAI Press, 2021, pp. 11939–11946.
[42] W. Min, B. W. Mott, J. P. Rowe, B. Liu, J. C. Lester, Player goal recognition in open-world digital games with long short-term memory networks, in: IJCAI, IJCAI/AAAI Press, 2016, pp. 2590–2596.
[43] N. Lesh, Adaptive goal recognition, in: IJCAI, Morgan Kaufmann, 1997, pp. 1208–1214.
[44] D. Bryce, J. Benton, M. W. Boldt, Maintaining evolving domain models, in: IJCAI, IJCAI/AAAI Press, 2016.
[45] T. Chakraborti, S. Sreedharan, Y. Zhang, S. Kambhampati, Plan explanations as model reconciliation: Moving beyond explanation as soliloquy, in: IJCAI, ijcai.org, 2017.
[46] D. Fahland, W. M. P. van der Aalst, Model repair - aligning process models to reality, Inf. Syst. 47 (2015) 220–243.
[47] A. Polyvyanyy, W. M. P. van der Aalst, A. H. M. ter Hofstede, M. T. Wynn, Impact-driven process model repair, ACM Trans. Softw. Eng. Methodol. 25 (2017) 28:1–28:60.
[48] V. Denisov, E. Belkina, D. Fahland, BPIC'2018: Mining concept drift in performance spectra of processes, in: 8th International Business Process Intelligence Challenge, 2018.
[49] B. Hompes, J. C. A. M. Buijs, W. M. P. van der Aalst, P. M. Dixit, H. Buurman, Detecting change in processes

using comparative trace clustering, in: SIMPDA, volume 1527 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2015.

[50] T. Chakraborti, Y. Zhang, D. E. Smith, S. Kambhampati, Planning with resource conflicts in human-robot cohabitation, in: AAMAS, ACM, 2016, pp. 1069–1077.

[51] Y. Jiang, H. Yedidsion, S. Zhang, G. Sharon, P. Stone, Multi-robot planning with conflicts and synergies, Auton. Robots 43 (2019) 2011–2032.

[52] G. Francés, M. Ramirez, Collaborators, Tarski: An AI planning modeling framework, https://github.com/aig-upf/tarski, 2018.

[53] M. Ramirez, N. Lipovetzky, C. Muise, Lightweight Automated Planning ToolKiT, http://lapkt.org/, 2015.

[54] M. Katz, S. Sohrabi, Reshaping diverse planning, in: AAAI, AAAI Press, 2020.

[55] Z. Su, T. Yu, N. Lipovetzky, A. Mohammadi, D. Oetomo, A. Polyvyanyy, S. Sardiña, Y. Tan, N. van Beest, Data-driven goal recognition in transhumeral prostheses using process mining techniques, in: ICPM, IEEE, 2023, pp. 25–32.