Towards Efficient Online Goal Recognition through Deep Learning

Lorenzo Serina
Università degli Studi di Brescia
Brescia, Italy
lorenzo.serina@unibs.it

Mattia Chiari
Università degli Studi di Brescia
Brescia, Italy
mattia.chiari@unibs.it

Alfonso Emilio Gerevini
Università degli Studi di Brescia
Brescia, Italy
alfonso.gerevini@unibs.it

Luca Putelli
Università degli Studi di Brescia
Brescia, Italy
luca.putelli@unibs.it

Ivan Serina
Università degli Studi di Brescia
Brescia, Italy
ivan.serina@unibs.it

ABSTRACT

Online goal recognition (OGR) is the task of understanding the intention of an agent as it executes a plan, recognizing its goal every time it performs a new action. This task is important in different contexts, such as applications of cyber-security and human-robot collaboration. An effective OGR system should (i) compute the correct goal of the agent as early as possible with respect to the agent's performed actions, and (ii) perform fast, given that, while the OGR system infers the agent's goal, the agent keeps executing the plan. In this paper, we propose a deep-learning approach to OGR based on Recurrent Neural Networks. The approach is implemented in a new system that learns to predict the goal of an agent acting in a given planning domain using a training dataset for the domain. We propose a method that exploits planning-related knowledge for designing a training dataset that is effective for the OGR and for improving the system performance. An experimental evaluation of our system on several benchmark domains shows that it performs generally better than the state-of-the-art in terms of accuracy and execution time, considering both the requirements (i) and (ii).

KEYWORDS

Online Goal Recognition; Deep Learning; Automated Planning

ACM Reference Format:
Lorenzo Serina, Mattia Chiari, Alfonso Emilio Gerevini, Luca Putelli, and Ivan Serina. 2025. Towards Efficient Online Goal Recognition through Deep Learning. In *Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Detroit, Michigan, USA, May 19 – 23, 2025*, IFAAMAS, 9 pages.

1 INTRODUCTION

The task of inferring the goals and intentions of an agent by observing its behaviour in an environment is defined as *goal recognition*. This task is the subject of many studies [17, 36] which analyse the problem from both the automated planning perspective [24, 26, 27] and the machine learning perspective [2, 6, 8, 9]. Typically, a goal recognition problem is specified as follows: assuming that the agent is executing a plan π to achieve a goal G^*, given a partial trace π' of possibly non-consecutive actions in π and a set $\mathcal{G}$ of possible goals for the agent containing G^*, the objective is to infer G^* from π' and $\mathcal{G}$. In the majority of the goal recognition studies, although the plan trace is often incomplete, it is typically assumed to be provided *offline*. That is, the trace is revealed after the agent has fully executed the plan and the candidate solution is generated only *once* for that trace. However, in many important domains, such as cyber-security and human-robot interaction, it is more useful to infer the goal of the agent during the plan execution (e.g., for early identification of possible damages of an attacker, or for providing collaborative support to an acting human). This task is called *online goal recognition* (shortly OGR) [38]. In OGR, it is important to consider that the trace is revealed *incrementally*, and the input of the goal recognition problem changes with it. Thus, a candidate solution is generated not just once as in offline goal recognition, but *every time a new observation is revealed*. Therefore, in OGR there are two important aspects to consider. The first, which is particularly relevant in security applications, is the *convergence* of the system [38], i.e., the ability to recognise the goal of the observed agent *as early as possible* with respect to the number of the performed plan actions. The second crucial aspect is goal-recognition time. In fact, while the system computes the solution, the agent keeps performing actions. Therefore, it could be useless to have a system with very high convergence that, in order to obtain such results, requires a substantial amount of time during which the agent can perform several additional actions.

For *offline* goal recognition, planning-based techniques can be rather effective, and the approaches in [26, 27] have obtained remarkable results. However, they need to use refined reasoning techniques based on automated planners, which can require even minutes to compute a solution. Moreover, although a faster approach has been proposed in [24], the best performance in terms of both accuracy and recognition time has been obtained by machine learning models, and in particular by GRNet [8].However, learning approaches can be sensitive to their training data, whereas symbolic approaches are more robust and general.

In this paper, we investigate OGR starting from GRNet, which we adapt to make it suitable for this problem. Next, we present a new approach called CLERNet (Causal Link Enhanced Recurrent Network) exploiting. CLERNet frames OGR as a many-to-many

Proc. of the 24th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2025), Y. Vorobeychik, S. Das, A. Nowé (eds.), May 19 – 23, 2025, Detroit, Michigan, USA. © 2025 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

sequence classification task performed by a Recurrent Neural Network. The network iteratively receives as input the incremental trace, and tries to predict the agent's goal every time a new action is executed. Each prediction is generated considering not only the last action executed, but also the contribution of all the actions executed before by the agent. To train the neural network, we propose two algorithms for creating an effective labelled dataset containing planning knowledge. Specifically, our system exploits *causal links* [21, 34] to identify which actions are necessary to achieve a goal, and uses the problem's initial state and mutually exclusive (*mutex*) relations among predicted facts. We experimentally evaluate GRNet and CLERNet on several planning benchmark domains, for which we created the datasets and trained domain-specific neural networks. Our results show that our approach obtains better performance in terms of convergence and recognition time compared to the state-of-the-art [37].

2 PRELIMINARIES

In this section we provide a formalisation of classical planning problems and online goal recognition problems in such a context.

Classical Planning. AI Planning involves extracting a sequence of actions (also called a plan) whose execution transforms a given initial state of the environment into a new state that satisfies the desired goal state [12]. In the branch of planning called Classical Planning, the environment is deterministic and the initial state and goal are fully known. The simplest classical planning language currently in use is STRIPS [10], a language based on boolean variables. In STRIPS, the boolean variables that compose a state of the environment are called *facts*, *fluents* or *atoms*. A planning problem in STRIPS is represented by a tuple $\Pi = \langle F, I, A, G \rangle$ where: F represents the set of all possible fluents or propositions of interest; $I \subseteq F$ represents the initial state; A represents the set of all possible actions and $G \subseteq F$ represents the goal. The actions $a \in A$ are represented by three sets of atoms over F called Add ($Add(a)$), Delete ($Del(a)$) and Precondition ($Pre(a)$) lists. The Add list describes the atoms that a makes true, the Delete list describes the atoms that a makes false and the Precondition list describes the atoms that must be true in order for the action a to be executed. A state s in STRIPS is a subset of F, with the meaning that if $f \in s$, then f is *true* in s, *false* otherwise. An action a is applicable in s if $s \models Pre(a)$, and the application of the action in s yields the state $s' = (s \backslash Del(a)) \cup Add(a)$. We indicate with $s' = s[a]$ the state resulting from applying the action a in s. A plan π for a planning problem Π is a sequence of actions $\pi = \langle a_1, \ldots a_n \rangle$ in Π; the plan π is a solution for Π iif there exists a sequence of states $\langle s_1, \ldots, s_n \rangle$ such that $s_1 = I$, and $\forall i \in \{1, \ldots, n\}$ we have that $s_i \models Pre(a_i)$, $s_{i+1} = s_i[a_i]$, and $s_{n+1} \models G$.

Given a plan π, it is possible to identify the set of *causal links* within π, representing the causal relationships between actions [21, 34]. These links indicate that the effect of one supporting action is necessary for the successful execution of another consumer action, without being threatened by any other action in the plan. Causal links are widely used in Partial Order Planning [39], or in explaining the behaviour of executed plans [32]. In our context, plans are sequences of actions rather than partially ordered sets of actions, as in partial-order planning. With sequential plans,the computation of causal links can be performed in polynomial time [5].

Online Goal Recognition. Our approach, as many others in the literature, belongs to the "goal recognition over a domain theory" field [24, 27, 36], into which the available knowledge consists of an underlying model of the agent's behaviour and its environment. This model typically represents the agent/environment states and the set of actions A that the agent can perform; typically, it is specified by a planning language such as PDDL. An *instance of the GR problem* in a given domain is then specified by: an initial state I of the agent and environment ($I \subseteq F$); a sequence $O = \langle o_1, .., o_n \rangle$ of observations ($n \geq 1$), where each o_i is an action in A performed by the agent, and a set $\mathcal{G} = \{G_1, .., G_m\}$ ($m \geq 1$) of possible goals of the agent, where each G_i is a set of fluents over F that represents a partial state. The observations form a trace of the full sequence π of actions performed by the agent to achieve a goal G^*.

We refer to *Offline Goal Recognition* when (i) the observation trace consists of possibly non-consecutive actions in π, ordered as they appear in π; and (ii) the candidate solution is generated once for a given problem. Whereas, following the assumptions stated in [38], we refer to *Online Goal Recognition* (OGR) when (i) the observation trace is complete, revealed incrementally, and consists of a prefix of π; and (ii) the candidate OGR solution is generated every time a new action is provided. Therefore, starting from the first action in π, for each new action executed by the agent, such action is added to the observation sequence and a candidate OGR solution is generated. Finally, there is no prior knowledge on the length of the sequence π; in other words, there is no information on whether the latest observation received is the final one.

Example 1. *As a very simple running example, we will use an OGR instance in the well-known* BLOCKSWORLD *domain. In this domain, one agent has the goal of building one or more stacks by moving only one block at a time. More specifically, in this domain there are four types of actions:* `Pick-Up` *a block from the table,* `Put-Down` *a block on the table,* `Stack` *a block on top of another one, and* `Unstack` *a block that is on another one. In this example, we assume that our OGR instance involves at most 4 blocks. In* BLOCKSWORLD *there are five types of fluents (predicates):* `On`*, which has two blocks as arguments, plus* `Clear`*,* `Holding` *and* `On-Table` *that have one argument and* `Handempty` *with no arguments. Subsequently, the fluent set F consists of 25 propositions. We then assume that the goal set $\mathcal{G}$ of the instance example consists of two candidate goals $G_1 =$ ⟨(`On Block_B Block_A`), (`On Block_D Block_C`)⟩ (which is the correct goal of the agent) and $G_2 =$ ⟨(`On Block_D Block_A`),(`On Block_B Block_C`)⟩ (which is another hypothetical goal). The plan executed by the agent, which corresponds to the complete sequence of observations $O = \langle o_1, \ldots, o_6 \rangle$ is made by 6 actions:* ⟨(`Unstack Block_A Block_C`), (`Put-Down Block_A`), (`Pick-Up Block_B`), (`Stack Block_B Block_A`), (`Pick-Up Block_D`), (`Stack Block_D Block_C`)⟩.

3 RELATED WORK

Although offline goal recognition has been extensively addressed through planning techniques [17, 24, 26, 27, 30, 31], matching techniques relying on plan libraries (e.g., [19]) and process mining techniques [33], online goal recognition has received less attention from the artificial intelligence community.

For offline goal recognition, the works in [26, 27] identify the goal and the plan by computing, for all the goals in the hypothesis set, an optimal plan from the initial state to the goal and an optimal plan that complies with the observations O from the same initial state to the same goal. Although this approach is formally exact, its major drawback is that it requires checking all candidate goals, with several calls to the planner, which could take a long time, limiting its applications for a time-sensitive task such as OGR. A faster planning-based approach called LGR has been introduced in [22, 24]. This approach exploits landmarks [13], i.e. properties or actions that cannot be avoided to achieve a goal. However, unlike the works in [26, 27], LGR does not have any formal guarantee of the correctness of the result, except for 100% of the plan.

For OGR, the work in [37] proposes an approach to solving instances in continuous domains. This approach, called *mirroring*, consists of the use of a planner to generate recognition hypotheses for a problem and all the possible goals. Then, it confronts such hypotheses with the actions observed to get a similarity measure used to identify the goal. Although this approach reaches very good results, there is an important disadvantage. In fact, to match the hypotheses with the actions, it is necessary to calculate several plans with a planner, resulting in a very long execution time on discrete domains. Therefore, in [38] the mirroring approach is extended with heuristic and landmarks (as in [22, 24]). The work in [38] currently represents the state-of-the-art for online goal recognition. In this paper, we will show how combining planning knowledge and deep learning can achieve better results with respect to [38].

More recently, goal recognition has been addressed using reinforcement learning [2, 9]. The work in [2] presents GRAQL, which is based on Q-learning. Given a GR instance, the approach learns a Q-table for each goal and computes the distance between the observed sequence and these policies to infer the candidate goal. The major difference between this work and ours is that GRAQL trains several specific RL models for each instance, whereas we train a general deep learning model that can solve many different instances. A similar problem-specific approach has been proposed in [9] but focusing on images and path-planning tasks which are not the subject of our work. Among the machine learning and deep learning approaches introduced for goal recognition, most are either application-focused [18, 20, 35] or problem-focused [6, 15]. Other approaches, like [3, 8], are designed to handle a wider set of problems. In [3], a LSTM(Long Short Term Memory) is used for filling missing observations in offline plan recognition task, trying to reconstruct the complete sequence of states reached by the agent. Since our formulation of OGR assumes a gap-free observation sequence, this approach becomes highly inefficient. Some OGR approaches with partial observability have been proposed in [23, 25], but this scenario is outside the scope of the present work. The work in [8] presents a LSTM network, GRNET, capable of solving many offline goal recognition problems in a discrete planning domain. Given the promising results, in this paper we perform the necessary steps to adapt GRNET and evaluate it in the context of OGR. Moreover, we propose a further improvement of such approach by incorporating planning knowledge, and in particular, causal links.

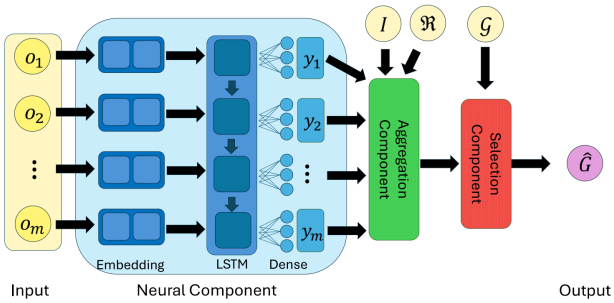

Figure 1: Architecture of CLERNET. Each observed action is processed by the Neural Component (in blue), which encodes the observations by embedding vectors, feeds them to an LSTM network and then to a feed-forward layer which, at each step, predicts the agent's goal scores (y_i). These outputs are then combined together with the initial state (I) and the information related to mutually exclusive fluents ($\mathcal{R}$) by the Aggregation Component (in green). Finally, the Selection Component returns the predicted goal $\hat{G}$ by processing the aggregated score and the set of candidate goals $\mathcal{G}$ (in red).

4 METHODOLOGY

In this section, we present CLERNET (*Causal Link Enhanced Recurrent Network*), our deep learning model for OGR. First, we describe its architecture how it incorporates planning-based information. Next, we describe the approaches based on causal links that we designed to create an effective training set.

4.1 Architecture of CLERNET

Following a similar approach to the one presented in [8], we structure our online goal recognition task as a multi-class classification on a sequential input. Therefore, we also chose a Long Short-Term Memory Network (LSTM) to process the sequence of actions observed. However, an important first difference has to be highlighted: GRNET is originally an offline goal recognition system, thus its neural network receives in input the observation trace of the plan and provides only one prediction after elaborating it in its entirety, in a task defined as a *many-to-one* sequential task; on the contrary, CLERNET receives in input one action at a time and, after every action, provides a new prediction (*many-to-many* sequential task). Therefore, considering an observation trace composed by m actions, the LSTM provides m different predictions. This approach enable us to both leverage the sequential nature of the observation trace and to combine these single predictions to produce a final prediction that accounts for all the agent's actions over time.

Our architecture, CLERNET, is depicted in Figure 1. It consists of three main parts: the *Neural Component* (in blue), the *Aggregation Component* (in striped green) and the *Selection Component* (in grid red). The input of the first component is an online goal recognition instance, and its output is a prediction vector for each input observation $o_i \in O$. Each prediction vector (y_i) is a vector of N components, one for each proposition in $F_G \subseteq F$, with a value in $[0, 1]$, where F_G is the domain fluent set that can appear in any goal of $\mathcal{G}$ for any GR instance in the domain. The Aggregation component takes as input the score vectors generated by the Neural Component, the initial

Input: The list of prediction vectors $Y = \langle y_i, \ldots, y_n \rangle$, a set mutually exclusive fluents $\mathcal{R}$, the initial state of the problem I and three numeric constants α, τ_1 and τ_2
Output: An aggregated prediction vector a

1 $a = [\ ]$
2 **for** $j \in |F_G|$ **do**
3 **if** $F_G[j] \in I$ **then** $a[j] = \alpha$
4 **else** $a[j] = 0$
5 **for** $y_i \in Y$ **do**
6 $t = \{j \mid y_i^j > \tau_1\}$
7 $mutex = \{\}$
8 **for** $j \in t$ **do** $mutex.add(\mathcal{R}(j) \setminus t)$
9 **for** $m \in mutex$ **do** $a[m] = 0$
10 **for** $j \in y_i$ **do**
11 **if** $y_i^j < \tau_2$ **then** $y_i^j = 0$
12 $a = a + y_i$
13 **return** a

Algorithm 1: Aggregation Component

state I of the agent and the information related to which fluents in F_G are mutually exclusive. Then, it calculates a unique aggregated score that summarizes all this information. Finally, the Selection Component receives in input the aggregated score and the set of candidate goals $\mathcal{G}$, and it outputs the predicted goal $\hat{G}$. Please note that the Neural Component is trained once for each domain and the overall architecture can be used for every online GR instance over F_G. On the other hand, both the Aggregation Component and the Selection Component are deterministic and do not require any training. In the remaining part of this section, we describe the main structure and behaviour of each component.

Neural Component. The Neural Component has the following structure. First, an embedding layer [4] transforms each input observation o_i into a vector e_i of real numbers. The index of each observation is simply the result of an arbitrary order of the set of all possible actions that can be observed in the domain, which is defined only once for each domain during the initial pre-processing phase. Then an LSTM layer processes the sequence of observations. The output of each cell is then passed to a time-distributed feed-forward layer, which has $N = |F_G|$ output neurons with a *sigmoid* activation function. Note that we don't use a specific feed-forward layer for each cell with a different weight matrix, but we use the same feed-forward layer (with the same weight matrix) for each cell. This layer provides a prediction for each observation we receive in input. Therefore, considering the prediction made after seeing the i-th observed action o_i, the output of the j-th neuron y_i^j corresponds to the score associated to the j-th fluent f_j (fluents are lexically ordered), and the activation value of y_i^j gives a score for f_j belonging to the agent's goal.

We used the binary cross-entropy as loss function for training the Neural Component. All hyperparameters, including the dimension of all the neural layers, are selected using the Bayesian optimisation approach provided by the Optuna framework [1]. Of the training set, 80% of it is used to train the network, wherare as the remaining 20% is used in this optimization.

Input: Planning Problem $\Pi = \{I, G, A\}$ and a totally ordered solution plan $\pi = \langle a_1, ..., a_n \rangle$ for Π
Output: A label associated to each action in π

1 $Goals = G$
2 $\mathcal{CL}_\pi = \{(a_i \xrightarrow{e} a_j) \mid i > j, e \in Add(a_i), e \in Pre(a_j), \nexists k \ s.t. \ i < k < j, \ e \in Del(a_k)\}$
3 **for** $i = n$ *to* 1 **do**
4 **if** $Goals \cap Add(a_i) \neq \emptyset$ **then**
5 $a_i.label = Goals \cap Add(a_i)$
6 $Goals = Goals \setminus Add(a_i)$
7 **else**
8 $acts(a_i) = \{a_j \in A | (a_i \xrightarrow{e} a_j) \in \mathcal{CL}_\pi\}$
9 $a_i.label = \bigcup_{b \in acts(a_i)} b.label$
10 **end**
11 **end**

Algorithm 2: Cumulative labelling strategy

Aggregation Component. The Aggregation Component aggregates the outputs of the previous component with the initial state I and with the information related to mutually exclusive fluents in F_G, following Algorithm 1. Consider providing a prediction after having observed the first m actions executed by the agent $\langle o_1, o_2 ... o_m \rangle$, the Neural Component provides the output $Y = \langle y_1, y_2, ..., y_m \rangle$, into which each fluent f_j has a score computed by the corresponding neuron j obtained after processing incrementally each observation o_i (y_i^j, with $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, N\}$).

From line 1 to 4, the aggregation vector a is initialized. a has the same structure as the other y_i and follows the same lexical ordering used in the Neural Component: it is a vector of size $|F_G|$ that has value α in position j if fluent f_j is in the initial state of the goal recognition problem and 0 otherwise. Eventual fluents that appear in the initial state but are not in F_G are not considered. Next, a **"for"** cycle loops over the predictions of the network. For each prediction y_i, at line 6 the algorithm identifies the index of the fluents that the network predicts belonging to the goal and stores them in t; we consider fluents predicted as belonging to the goal set if their score exceeds τ_1. The algorithm then computes the indexes of those fluents which are mutually exclusive to those in t and that do not appear in t, and sets them to 0 in the aggregation vector a (lines 7-9). The idea behind this procedure is to include domain knowledge to correct errors made when considering only the first observations. After seeing just a few actions, in fact, it is probable that some fluents are not predicted correctly. However, when provided with additional actions as input, the neural network may alter its predictions, potentially contradicting previous ones. In our procedure, we prioritise predictions based on the most recent information, overriding any prior conflicting predictions. In order to simplify the calculations and to avoid noise propagation errors, all scores lower than a threshold τ_2 are set to zero. From line 10 to line 12, scores predicted in y_i are then added to the aggregation vector a. Finally, the aggregation vector a is returned. In our experimental evaluation, we set α to 0.1, τ_1 to 0.4 and τ_2 to 0.05

Selection Component. The Selection Component performs an evaluation of the candidate goals in $\mathcal{G}$ of the OGR instance, using the aggregated vector a. To choose the most probable goal in $\mathcal{G}$ (solving the multi-class classification task corresponding to the

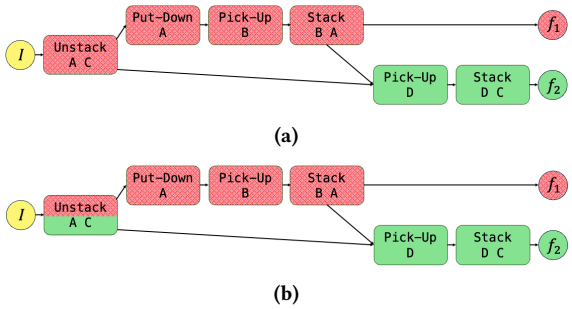

Figure 2: Visualisation of the Proximity (a) and Cumulative (b) labeling strategies for our example. Each rectangle represents an action and their colour represents which goal fluents are the label assigned to each action (red for f_1, green for f_2). Each line represents a causal link.

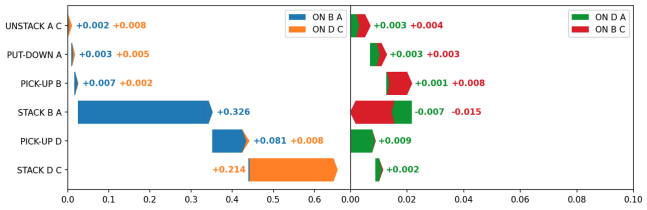

Figure 3: Visualization of the incremental score calculated by the Neural and the Aggregation components of CLERNet for our running example. On the y-axis we have the actions progressively seen by the system. On the x-axis, arrows shows the contribution of each action to the score of each candidate goal and its fluents (G_1 on the left, G_2 on the right).

OGR instance), we designed a simple score function that indicates how likely it is that G is the correct goal, according to the previous components. This score is defined as $S(G) = \sum_{f \in G} a[f]$ where $a[f]$ is the output of the aggregator component for f of the current OGR instance. For each candidate goal $G \in \mathcal{G}$, we consider only the output neurons that have associated facts in G. By adding only these predicted values, we derive an overall score for G being the correct goal, and we select the element with the highest score in $\mathcal{G}$.

4.2 Dataset Creation and Causal Link Labelling

Our training set for the Neural Component of CLERNet, consists of many couples (O, G^*), with O the complete sequence of observed actions belonging to a plan π and G^* the hidden goal satisfied by that plan. In order to create such dataset, we exploited the problems and the solution plans (and therefore their dimensions in terms of number of actions, goal fluents and possible goals) released in [8]. For solving such problems and obtaining the plans (and therefore the sequence of observations O), the authors of [8] used the LPG [11] planner, which allows the specification of the number of different solutions required for the problem it resolves. In our dataset, we included 4 sub-optimal solutions. Given that we address OGR as a many-to-many sequential task, we need a label for each observed action in input. Although the simplest technique could be to replicate the label corresponding to G^* for all observed actions, this has not been proven to be effective. Moreover, different actions can contribute to different goal fluents. Consider the problem described in Example 1. The action of picking up one of the blocks that builds the first tower is certainly relevant for that tower, but it has a very limited impact on the second tower. This kind of information is provided by *causal links* in the solution plan and can be very important for our model to learn the connections among the actions executed by an agent and its goals. Therefore, in order to include causal links in the training process of our Neural Component, we designed two ways to assign a label for each observation $o_i \in O$: the Cumulative and the Proximity labelling strategies.

Algorithm 2 describes the Cumulative labelling strategy. It receives as input a problem Π and a corresponding solution plan π. Without loss of generality, we assume that π is totally ordered and hence each action of π is associated to an integer time step; in the case of partially ordered plans, the system can simply derive a total order from them. At line 2, all causal links are computed; then the **for** cycle (line 3) examines in reverse order all the actions of π to identify for each of them a set of label fluents that can be used during the training phase. In particular, if some additive effects of the current action are included in *Goals* then the label of this action corresponds to these facts (line 5), and we remove them from *Goals*. Otherwise, we identify all the actions with a precondition supported by the current action (line 8), and we define the label of this action as the union of the labels of the previously identified actions; note that these labels correspond to goal facts already satisfied by actions causally ordered with the current action.

The algorithm for the Proximity selection procedure selects, at line 8, only the causally ordered action closest to the current one, i.e $acts(a_i) = argmin_j\{a_j \in A | (a_j \xrightarrow{e} a_j) \in \mathcal{CL}_\pi\}$, and in this case, the label of the current action corresponds to the label of the closest action in the causal links.

Example 2. *Considering the problem described in Example 1, in Figure 2 we show the label assigned by the Proximity (2a) and Cumulative (2b) strategies. In the figure, we show the initial state I, the actions progressively observed by the OGR system (represented as rectangles), and the fluents (represented as circles) belonging to the correct goal G_1: f_1 =* `(On Block_B Block_A)` *(in red) and f_2 =* `(On Block_D Block_C)` *(in green). Each line between two actions or between an action and a fluent represents a causal link. Exploiting this information, we can label each action with one or more goal fluents that they achieve. If a causal link involves a goal fact, we assign that fluent as the label of the supporter action. For example, since the action* `(Stack Block_B Block_A)` *is the supporter action of f_1, we assign f_1 as its label (in red). Recursively, the actions that have a causal link with* `(Stack Block_B Block_A)` *(such as* `(Pick-Up Block_B)`*) have f_1 as label too. The same happens with those that have a causal link with* `(Pick-Up Block_B)` *and so on. The main difference between the Cumulative and Proximity strategies is on how labels are assigned to those actions that have a causal link leading to two different fluents. This is the case of* `(Unstack Block_A Block_C)` *which allows the agent to put* `Block_A` *on the table (*`(Put-Down Block_A)`*) and stack* `Block_B` *on top of it in order to achieve f_1, as well to remove an object over* `Block_C`*, allowing the agent to pick-up* `Block_D` *to put on top of* `Block_C` *for achieving f_2. With Cumulative (Figure 2b) both fluents are assigned*

Table 1: Results of ORL [38], GRNet [8] and CLERNet for six planning domains. For each system, we consider the Ranked First (RF), the Convergence (CV) metrics and the execution time (T, in *ms*). Best results are marked in bold, second best underlined.

Domain	ORL			GRNet			Cumulative			Proximity		
	RF	CV	T	RF	CV	T	RF	CV	T	RF	CV	T
blocksworld	38.9	36.3	580	41.5	36.4	11	43.6	40.6	11	**46.6**	**44.2**	11
depots	46.6	**44.9**	987	47.3	40.6	4	49.1	42.6	4	**50.6**	44.7	4
driverlog	50.9	48.3	897	57.6	48.7	4	**59.8**	**55.3**	5	59.5	53.8	4
logistics	49.5	48.6	1118	**58.5**	**51.7**	4	56.8	49.8	4	55.0	**51.7**	4
satellite	**72.5**	**69.8**	1260	66.8	60.3	7	70.2	68.0	7	70.1	68.5	7
zenotravel	57.7	56.8	649	69.2	66.3	6	75.2	71.0	5	**75.4**	**71.7**	5
average	52.9	51.1	920	57.1	50.9	6	59.4	54.8	6	**59.8**	**56.0**	6

as a label. Instead, with Proximity only f_1 is assigned because the action `Put-Down A`*, which has label f_1, is closer to* `(Unstack Block_A Block_C)` *than action* `(Pick-Up D)`*, which has label f_2.*

The labels are structured as binary vectors. In our example, that involves at most 25 *goal fluents, an index between* 0 *and* 24 *is assigned to each fluent and a vector of dimension* 25 *is created. Let's assume that f_1 has index* 8*. The action* `(Put-Down Block_A)` *and all others connected with f_1 will have as a label a vector with all zeroes except a* 1 *in position* 8*. With the Cumulative strategy, if an action has two or more labels, the vector will contain a* 1 *for each label.*

An intuition to how our system works is given in Figure 3, which shows the scores (calculated by the Neural Component and the Aggregation component) for each candidate goal. On the left, we show the score calculated for $G_1 = \langle$ `(On Block_B Block_A)`*,* `(On Block_D Block_C)` *$\rangle$ (which is the correct goal of the agent), whereas on the right we show the score of $G_2 = \langle$* `(On Block_D Block_A)`*,* `(On Block_B Block_C)` *$\rangle$. On the y-axis we have the initial state of our OGR problem and the actions progressively seen by our system. On the x-axis, we have the score given by the Aggregation Component for both fluents. As we explained in Section 4.1, this score is calculated incrementally, therefore we show the progressive contribution of each action to the prediction. We can see that the first three actions do not provide a valuable contribution to any of the goals. Instead,* `(Stack Block_B Block_A)` *makes the Neural Component and the Aggregation Component change their predictions and lean towards G_1, recognising that such action is fundamental for* `(On Block_B Block_A)`*. Instead, this action has a negative effect on the score of G_2. This is due to the fact that* `(On Block_B Block_A)` *is predicted as true (i.e. with a score greater than τ_1). This causes the fluents* `(On Block_D Block_A)` *and* `(On Block_B Block_C)` *in G_2, which are mutually exclusive with* `(On Block_B Block_A)`*, to be set to 0. As expected, the remaining actions of the plan (in particular,* `(Stack Block_D Block_C)`*) increment the score of G_1, which will be correctly chosen by the Selection Component.*

5 EXPERIMENTAL EVALUATION

For the experimental evaluation, we choose six different discrete planning domains with unit action costs [14, 16]: blocksworld, depots, driverlog, logistics, satellite, zenotravel. For each considered domain, we created a training set with 55,000 instances using the labelling strategies we described in Section 4.2 and the goal recognition problems of [8]. As a test set, we used the set of 6,000 goal recognition instances published and used in [8], which were generated with the planner lama [28].

To evaluate our models, we use different metrics. First, we consider the prediction accuracy for different portions of the plan. We examine it ranging from 10% to 100% of actions, with a 10% step. Furthermore, we compute the *Ranked First* (RF) and *Convergence* (CV) metrics, which are standard metrics for OGR introduced in [38]. For a single OGR instance, RF is defined as the number of predicted goals (one for each observation) correctly identified, divided by the total number of observations. The CV metric expresses how early our model can predict the correct goal, and it is calculated similarly to RF; this metric considers a goal predicted correctly not only if it is the correct goal, but also if the system does not predict another goal from that point onwards. More intuitively, consider the OGR instance in Examples 1 and 2 made by 6 observations. A system predicting G_1 after observing the first action, G_2 after observing the second and third actions, and predicting G_1 from the fourth action obtains *RF* equal to $(1+0+0+1+1+1)/6 = 4/6$ and *CV* equal to $(0+0+0+1+1+1)/6 = 3/6$. Considering the whole test set, RF and CV are defined as the average of the metrics across all test instances. We also include the average prediction/execution time (T), which does not include the training time (approximately 1 hour on CPUs). Nonetheless, the networks can be applied to multiple instances in the domain.

We compare the deep learning-based approaches (GRNet and CLERNet) first with the state-of-the-art methods based on reasoning (ORL [38]) and on reinforcement learning (graql [2]). For applying GRNet to OGR, considering a plan composed by m actions, we create m action traces, starting from one containing only the first action executed by the agent and adding the subsequent actions one at a time. This poses a significant difference with the system application in [8] as, in that context, GRNet was applied to *offline goal recognition*, and thus it provided a single prediction for each incomplete trace. For the comparison with graql, we adopt two additional test sets in the blocksworld domain. Firstly, we use the set of 10 problems presented in [2] (bw_rl), using the complete observation trace to generate the OGR instances. Secondly, given the rather small number of blocks used in these instances (up to 6), we also randomly selected 10 problems form our testset to evaluate the performance on more complex instances (bw_small).

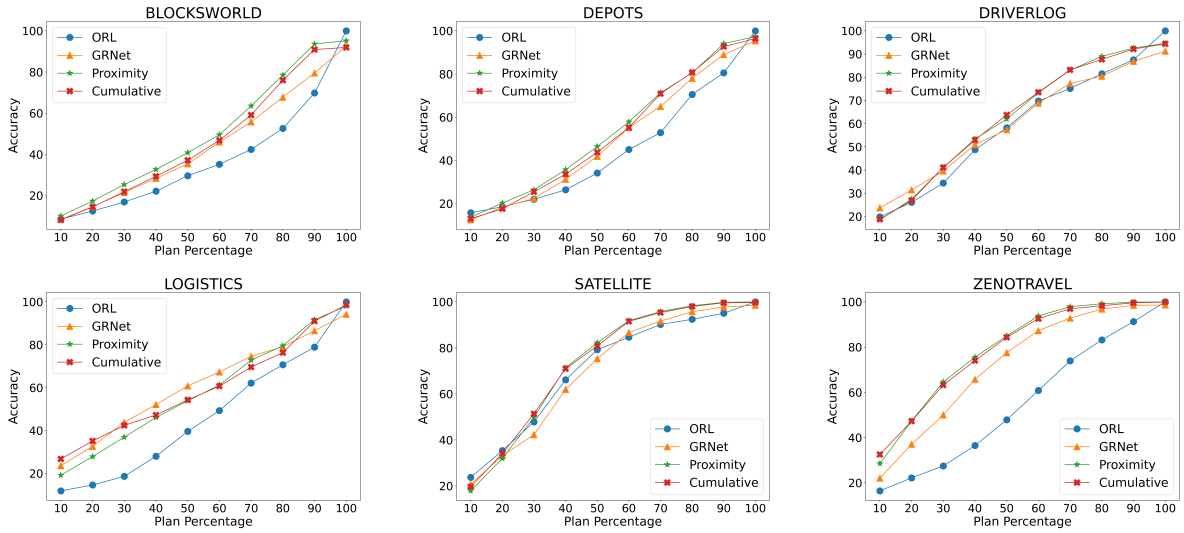

Figure 4: Performance of CLERNet with Proximity (in green) and Cumulative (in red) labeling strategies for each considered domain, compared with ORL (in blue) and GRNet (on range) in terms of accuracy (on the y-axis). On the x-axis, we indicate the percentage of actions in the plan observed and given in input.

5.1 Comparison with ORL

The results of our comparison among CLERNet, GRNet and ORL, in terms of RF, CV and T, are available in Table 1.

First, we can observe that adapting GRNet to OGR provides better results w.r.t ORL in terms of RF in all domains except satellite. The major improvements are reported in driverlog (57.6 points versus 50.9), logistics (58.5 versus 49.5) and zenotravel (69.2 versus 57.7). In terms of CV, the results of GRNet and ORL are comparable. In fact, we can see an important improvement in zenotravel (9.5 points) and in logistics (3.1 points), but a marked worsening in depots (4.3 ponts) and satellite (9.5 points). On average, GRNet obtains a higher RF (57.1 versus 52.9) and a comparable CV (50.9 versus 51.1). However, the most important difference between these two approaches is in terms of time. In fact, while GRNet takes around 6 ms on average to compute the prediction after each observation, ORL takes about 150 times more (920 ms). Given that in the OGR context recognition time can be fundamental, as the agent keeps executing actions while the system is trying to understand in goal, GRNet overall performs better than ORL.

An even more important difference can be seen comparing ORL to our new model, CLERNet, considering both the Proximity and the Cumulative labeling strategies. As it can be seen in Table 1, CLERNet obtains better results in all domains except satellite in terms of RF. The most important improvements can be seen in zenotravel (75.4 obtained by Proximity) and driverlog (59.8 obtained by Cumulative, versus 50.9). In terms of CV, CLERNet performs better in all domains except satellite and slightly in depots, into which ORL obtains a CV of 44.9 and CLERNet with the Proximity strategy obtains a very similar 44.7. Analysing satellite, we can see a slightly less marked worsening of the performance (72.5 obtained by ORL versus 70.2 by Cumulative and 70.1 by Proximity, in terms of RF, and 69.8 versus 68.0 and 68.5 in terms of CV) with respect to the worsening that can be seen with GRNet. On average, our approach performs better than ORL by about 5 points both in terms of RF and CV. Similarly to what previously observed for GRNet, we remark the important difference between CLERNet and ORL in terms of time (6*ms* versus 920*ms*).

Finally, we compare the performance of GRNet and CLERNet. In terms of RF and CV, CLERNet obtains better performance for all domains except logistics (58.5 versus 56.8 obtained by Cumulative and 55.0 obtained by Proximity in terms of RF). The most important improvements can be seen in zenotravel (about 5 point of RF and CV) and blocksworld (44.2 obtained by Proximity versus 36.4 of GRNet in terms of CV). On average, we improve by more than 2 points in terms of RF. In terms of CV, Proximity obtains 5 points more w.r.t to GRNet. Since they are both LSTM based, CLERNet and GRNet have similar recognition time.

In terms of accuracy (see Figure 4), we observe the same trend we observed for RF and CV. In almost all cases, we can see that ORL (in blue) obtains a lower accuracy for almost all the plan percentages. However, some peculiar exceptions are satellite and depots with 10% of the plan. In such cases, the best performance is obtained by ORL. This is probably due to the landmark computation, which includes fundamental information that cannot be deducted by the small amount of observations in input. This slight difference of performance for depots is the main motivation behind the better CV, whereas in general GRNet and CLERNet obtain better performance. Comparing the deep-learning based approaches, we can see how CLERNet (green and red lines) obtains a better accuracy w.r.t. to GRNet even with small percentage of plans. A notable exception is logistics, in which GRNet obtains higher results than our approaches, especially with lower plan percentages. With more information (from 70%), both approaches perform very well. The reported improvements are statistically significant according to the Friedman test and the Nemeny Post-hoc test. The results on these

Table 2: Comparison in terms of RF, CV and T (in seconds) between GRAQL and CLERNET with PROXIMITY labels on BW_RL and BW_SMALL datasets. For each dataset, we report the min and max number of blocks (# Bl) and the goal set size ($|\mathcal{G}|$). the For GRAQL, we report in brackets the average time needed to learn the policies for each considered problem.

Domain	# Bl	$\lvert\mathcal{G}\rvert$	GRAQL			PROXIMITY		
			RF	CV	T	RF	CV	T
BW_RL	[5, 6]	4	82.9	82.9	0.08 (7)	48.8	47.5	0.02
BW_SMALL	[8, 11]	20	0.0	0.0	22.0 (4420)	48.8	32.5	0.01

tests confirm that there is a significant difference between CLERNET (both considering PROXIMITY and CUMULATIVE) and ORL, and between CLERNET and GRNET, with the only exception in SATELLITE, for which the difference with ORL is not statistically significant.

Comparing our two labeling strategies, we can see that the results are quite similar, with an average difference of 0.4 for RF and about 1 for CV. There is however a notable exception: in BLOCKSWORLD, PROXIMITY performs better than CUMULATIVE, especially in terms of CV (44.2 versus 40.6). This is probably due to the high number of causal links that can be extracted from a plan in that domain, which complicates the learning process of the Neural Component the CUMULATIVE strategy.

5.2 Comparison with GRAQL

In this subsection, we compare CLERNET and GRAQL [2] considering both the BW_RL test set and the 10 problems in BW_SMALL.

GRAQL was implemented using the default configuration (i.e. 500 episodes and 10 seconds as time out limit for the optimal planner initialization). The results, in terms of RF, CV and time (T) are in Table 2. Considering BW_RL, GRAQL reaches excellent performance, with 82.9 in both RF and CV, whereas PROXIMITY obtains results similar to those given in Table 1 (48.8 of RF and 47.5 of CV). This notable difference in performance highlights the potential of RL-based methods. However, these results require a specific training procedure that computes multiple Q-Tables for each OGR instance. Thus, this problem-specific approach requires an additional computation time (7 seconds on average). For our approach, this time is not included in Table 2, as CLERNET does not require additional processing time for new instances once trained. Moreover, as we can see from the results on BW_SMALL, GRAQL has several limits in terms of scalability. In fact, we tested the same configuration used by GRAQL on BW_SMALL without obtaining meaningful results. Even testing several other configurations considered by the authors of GRAQL (10k and 30k episodes, time limit up to 15 minutes) did not produce any improvements (0 in terms of both RF and CV). As it can be seen in Table 2, increasing the size of the test problems (from up to 6 blocks to up to 11 blocks) results in policies that were insufficiently informed and deviated from the goal during execution. This may be attributed to the larger state space in the Q-table, given by the higher number of blocks considered. Moreover, BW_SMALL contains suboptimal plans whereas GRAQL contains optimal plans. As expected, increasing the number of goals led to a corresponding increase in the time required to compute the Q-Tables. Instead, CLERNET does not suffer these problems and it has very similar results in terms of RF, CV and a very similar execution time.

Table 3: Results of the ablation study in terms of RF and CV for PROXIMITY. *No* stands for no knowledge included, *Mutex* for including only mutually exclusive facts, I including only the initial state and *Both* including both Mutex and I.

Domain	No		Mutex		I		Both	
	RF	CV	RF	CV	RF	CV	RF	CV
BW	**47,1**	44,5	**47,1**	44,1	46,7	**44,7**	46,6	44,2
DEPOTS	45.8	42.0	49.0	43.7	47.1	42.6	**50.6**	**44.7**
DRIVERLOG	57.7	51.6	58.3	51.5	58.9	53.7	**59.5**	**53.8**
LOGISTICS	55.3	51.4	**55.5**	51.4	54.8	**51.7**	55.0	**51.7**
SATELLITE	**70.1**	68.3	**70.1**	68.2	**70.1**	**68.5**	**70.1**	**68.5**
ZENOTRAVEL	72.1	68.3	72.4	68.8	75.0	71.2	**75.4**	**71.7**

5.3 Ablation Study

In order to measure the contribution of the different sources of planning knowledge in our Aggregation Component, we conducted an ablation study considering the PROXIMITY labeling strategy; these results are in Table 3. As we can see in most of the cases, despite CLERNET performs well even without the inclusion of mutually exclusive fluents (Mutex) and the initial state (I), this knowledge helps the system to predict the correct goal, increasing the RF and CV metrics. The most important improvements can be seen for DEPOTS, in which performance increases from 45.8 to 49.0 in terms of RF including mutex facts and to 50.6 also including I. We observe good improvements also for ZENOTRAVEL, while for the other domains they are less significant. However, it is important to notice that all the models in the ablation study include a form of planning related knowledge given by the causal links exploited for training the Neural Component.

6 CONCLUSIONS AND FUTURE WORK

We have studied how deep-learning based models can be applied to the online goal recognition (OGR) task. We adapted and tested GRNET [8] and introduced CLERNET, which leverages valuable planning knowledge. Our experimental analysis shows that they can obtain better performance w.r.t. the state-of-the-art [38].

Regarding future work, we plan to test different labeling techniques and use information from the current state to refine the neural network's output, and improve accuracy. We will also consider the integration of a symbolic technique, as in [7], to deal with uncertain prediction and make CLERNET more robust. Moreover, a GPT-based approach, such as the one proposed in [29] for planning, will be studied in the context of goal recognition.

ACKNOWLEDGMENTS

This work was partially supported by project SERICS (PE00000014), project RIPER (No. 20203FFYLK), by project FAIR (B53C22003980006), under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU.

REFERENCES

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2623–2631.

[2] Leonardo Amado, Reuth Mirsky, and Felipe Meneguzzi. 2022. Goal recognition as reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 9644–9651.

[3] Leonardo Amado, Ramon Fraga Pereira, and Felipe Meneguzzi. 2023. Robust Neuro-Symbolic Goal and Plan Recognition. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, Brian Williams, Yiling Chen, and Jennifer Neville (Eds.). AAAI Press, 11937–11944. https://doi.org/10.1609/AAAI.V37I10.26408

[4] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *The journal of machine learning research* 3 (2003), 1137–1155.

[5] Pascal Bercher. 2021. A Closer Look at Causal Links: Complexity Results for Delete-Relaxation in Partial Order Causal Link (POCL) Planning. In *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling, ICAPS 2021, Guangzhou, China (virtual), August 2-13, 2021*, Susanne Biundo, Minh Do, Robert Goldman, Michael Katz, Qiang Yang, and Hankz Hankui Zhuo (Eds.). AAAI Press, 36–45. https://ojs.aaai.org/index.php/ICAPS/article/view/15944

[6] Daniel Borrajo, Sriram Gopalakrishnan, and Vamsi K. Potluru. 2020. Goal recognition via model-based and model-free techniques. *Proceedings of FinPlan 2020* (2020).

[7] Mattia Chiari, Alfonso Emilio Gerevini, Andrea Loreggia, Luca Putelli, and Ivan Serina. 2024. Fast and Slow Goal Recognition. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2024, Auckland, New Zealand, May 6-10, 2024*, Mehdi Dastani, Jaime Simão Sichman, Natasha Alechina, and Virginia Dignum (Eds.). International Foundation for Autonomous Agents and Multiagent Systems / ACM, 354–362. https://doi.org/10.5555/3635637.3662884

[8] Mattia Chiari, Alfonso Emilio Gerevini, Francesco Percassi, Luca Putelli, Ivan Serina, and Matteo Olivato. 2023. Goal Recognition as a Deep Learning Task: The GRNet Approach. In *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling, July 8-13, 2023, Prague, Czech Republic*. AAAI Press, 560–568.

[9] Zihao Fang, Dejun Chen, Yunxiu Zeng, Tao Wang, and Kai Xu. 2023. Real-Time Online Goal Recognition in Continuous Domains via Deep Reinforcement Learning. *Entropy* 25, 10 (2023), 1415. https://doi.org/10.3390/E25101415

[10] Richard E. Fikes and Nils J. Nilsson. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* (1971).

[11] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina. 2003. Planning Through Stochastic Local Search and Temporal Action Graphs in LPG. *J. Artif. Intell. Res.* 20 (2003), 239–290.

[12] Malik Ghallab, Dana S. Nau, and Paolo Traverso. 2004. *Automated planning - theory and practice*. Elsevier.

[13] Jörg Hoffmann, Julie Porteous, and Laura Sebastia. 2004. Ordered Landmarks in Planning. *J. Artif. Intell. Res.* 22 (2004), 215–278.

[14] Derek Long and Maria Fox. 2003. The 3rd International Planning Competition: Results and Analysis. *J. Artif. Intell. Res.* 20 (2003), 1–59.

[15] Mariane Maynard, Thibault Duhamel, and Froduald Kabanza. 2019. Cost-Based Goal Recognition Meets Deep Learning. *Proceedings of PAIR 2019* (2019).

[16] Drew V. McDermott. 2000. The 1998 AI Planning Systems Competition. *AI Mag.* 21, 2 (2000), 35–55.

[17] Felipe Meneguzzi and Ramon Fraga Pereira. 2021. A Survey on Goal Recognition as Planning. In *Proceedings of IJCAI 2021*.

[18] Wookhee Min, Bradford W. Mott, Jonathan P. Rowe, Barry Liu, and James C. Lester. 2016. Player Goal Recognition in Open-World Digital Games with Long Short-Term Memory Networks. In *Proceedings of IJCAI 2016*. IJCAI/AAAI Press.

[19] Reuth Mirsky, Roni Stern, Ya'akov (Kobi) Gal, and Meir Kalech. 2016. Sequential Plan Recognition. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*, Subbarao Kambhampati (Ed.). IJCAI/AAAI Press, 401–407.

[20] Zaolin Pan and Yantao Yu. 2024. Learning multi-granular worker intentions from incomplete visual observations for worker-robot collaboration in construction. *Automation in Construction* 158 (2024), 105184.

[21] J. Scott Penberthy and Daniel S. Weld. 1992. UCPOP: A Sound, Complete, Partial Order Planner for ADL. In *KR*. Morgan Kaufmann, 103–114.

[22] Ramon Pereira, Nir Oren, and Felipe Meneguzzi. 2017. Landmark-based heuristics for goal recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.

[23] Ramon Fraga Pereira, Francesco Fuggitti, Felipe Meneguzzi, and Giuseppe De Giacomo. 2024. Temporally extended goal recognition in fully observable non-deterministic domain models. *Appl. Intell.* 54, 11-12 (2024), 470–489. https://doi.org/10.1007/S10489-023-05087-1

[24] Ramon Fraga Pereira, Nir Oren, and Felipe Meneguzzi. 2020. Landmark-based approaches for goal recognition as planning. *Artif. Intell.* 279 (2020).

[25] Ramon Fraga Pereira, Mor Vered, Felipe Meneguzzi, and Miquel Ramírez. 2019. Online Probabilistic Goal Recognition over Nominal Models. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, Sarit Kraus (Ed.). ijcai.org, 5547–5553. https://doi.org/10.24963/IJCAI.2019/770

[26] Miquel Ramírez and Hector Geffner. 2009. Plan Recognition as Planning. In *Proceedings of IJCAI 2009*.

[27] Miquel Ramírez and Hector Geffner. 2010. Probabilistic Plan Recognition Using Off-the-Shelf Classical Planners. In *Proceedings of AAAI 2010*. AAAI Press.

[28] Silvia Richter and Matthias Westphal. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *J. Artif. Intell. Res.* 39 (2010), 127–177.

[29] Nicholas Rossetti, Massimiliano Tummolo, Alfonso Emilio Gerevini, Luca Putelli, Ivan Serina, Mattia Chiari, and Matteo Olivato. 2024. Learning General Policies for Planning through GPT Models. In *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024, Banff, Alberta, Canada, June 1-6, 2024*, Sara Bernardini and Christian Muise (Eds.). AAAI Press, 500–508. https://doi.org/10.1609/ICAPS.V34I1.31510

[30] Luísa R. A. Santos, Felipe Meneguzzi, Ramon Fraga Pereira, and André Grahl Pereira. 2021. An LP-Based Approach for Goal Recognition as Planning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*. AAAI Press, 11939–11946.

[31] Shirin Sohrabi, Anton V. Riabov, and Octavian Udrea. 2016. Plan Recognition as Planning Revisited. In *Proceedings of IJCAI 2016*. IJCAI/AAAI Press.

[32] Sarath Sreedharan, Tathagata Chakraborti, and Subbarao Kambhampati. 2021. Foundations of explanations as model reconciliation. *Artificial Intelligence* 301 (2021), 103558. https://doi.org/10.1016/j.artint.2021.103558

[33] Zihang Su, Artem Polyvyanyy, Nir Lipovetzky, Sebastian Sardiña, and Nick van Beest. 2024. Adaptive goal recognition using process mining techniques. *Eng. Appl. Artif. Intell.* 133 (2024), 108189. https://doi.org/10.1016/J.ENGAPPAI.2024.108189

[34] Austin Tate. 1977. Generating Project Networks. In *International Joint Conference on Artificial Intelligence*. https://api.semanticscholar.org/CorpusID:538251

[35] Nusrat Jahan Tithi and Swakkhar Shatabda. 2023. A Convolutional Neural Network for Goal Recognition. In *2023 26th International Conference on Computer and Information Technology (ICCIT)*. 1–6.

[36] Franz A. Van-Horenbeke and Angelika Peer. 2021. Activity, Plan, and Goal Recognition: A Review. *Frontiers Robotics AI* 8 (2021).

[37] Mor Vered, Gal A. Kaminka, and Sivan Biham. 2016. Online goal recognition through mirroring: humans and agents. In *Fourth Annual Conference on Advances in Cognitive Systems (Advances in Cognitive Systems)*, Kenneth Forbus, Tom Hinrichs, and Carrie Ost (Eds.). Cognitive Systems Foundation.

[38] Mor Vered, Ramon Fraga Pereira, Mauricio Cecilio Magnaguagno, Gal A. Kaminka, and Felipe Meneguzzi. 2018. Towards Online Goal Recognition Combining Goal Mirroring and Landmarks. In *AAMAS*. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2112–2114.

[39] Håkan L. S. Younes and Reid G. Simmons. 2003. VHPOP: Versatile Heuristic Partial Order Planner. *J. Artif. Intell. Res.* 20 (2003), 405–430. https://doi.org/10.1613/JAIR.1136