Knowledge and Information Systems manuscript No.
(will be inserted by the editor)

Cyclic Action Graphs for goal recognition problems with inaccurately initialised fluents

Helen Harman · Pieter Simoens

Received: date / Accepted: date

Abstract Goal recognisers attempt to infer an agent's intentions from a sequence of observed actions. This is an important component of intelligent systems that aim to assist or thwart actors; however, there are many challenges to overcome. For example, the initial state of the environment could be partially unknown, agents can act suboptimally and observations could be missing. Approaches that adapt classical planning techniques to goal recognition have previously been proposed but, generally, they assume the initial world state is accurately defined. In this paper, a state is inaccurate if any fluent's value is unknown or incorrect. Our aim is to develop a goal recognition approach that is as accurate as the current state of the art algorithms and whose accuracy does not deteriorate when the initial state is inaccurately defined. To cope with this complication, we propose solving goal recognition problems by means of an Action Graph. An Action Graph models the dependencies, i.e., order constraints, between all actions rather than just actions within a plan. Leaf nodes correspond to actions and are connected to their dependencies via operator nodes. After generating an Action Graph, the graph's nodes are labelled with their distance from each hypothesis goal. This distance is based on the number and type of nodes traversed to reach the node in question from an action node that results in the goal state being reached. For each observation, the goal probabilities are then updated based on either the distance the observed action's node is from each goal or the change in distance. Our experimental results, for 15 different domains, demonstrate that our approach is robust to inaccuracies within the defined initial state.

Keywords goal recognition · graph algorithm · cyclic graph · inferring intent · context aware

Department of Information Technology - IDLab,
Ghent University - imec,
Technologiepark 126, B-9052 Ghent, Belgium
E-mail: Helen.Harman@ugent.be · E-mail: Pieter.Simoens@ugent.be

1 Introduction

By observing the behaviour of an agent, artificially intelligent systems can attempt to determine the agent's intentions. Knowledge of an agent's intentions is essential in numerous application areas. These include computer games in which non-playable characters must adapt to players' actions [8]; intelligent user help for human-computer interaction scenarios [22, 23]; offering humans energy saving advice [33]; robot sports playing (e.g., table tennis [56]); interfering (and thus preventing) the intentions of computer network intruders [13, 37]; determine the location a human is navigating to (e.g., for airport security) [34], and to enable proactive robot assistance [11, 30, 31]. Rather than developing domain specific intention recognition algorithms, a symbolic representation of the world and agents' actions can be provided as input to non-domain specific algorithms [48, 55].

Intention recognition can be split into several categories, namely, activity recognition [32, 53, 57], plan recognition [13, 36, 51], and goal recognition [39, 44]. Our work falls under the category of goal recognition (GR), in which the aim is to label a sequence of observations (e.g., actions) with which goal the observee is attempting to reach. For instance, when provided with a sequence of move actions, GR methods will attempt to select (from a predefined list) which location the agent is intending to reach. For the Kitchen domain by Ramírez and Geffner [44], the sequence of observed actions includes taking different items and using appliances (e.g., a toaster), and the returned classification indicates if the observee is likely to be making breakfast, dinner or a packed lunch. Goal and plan recognisers operate on discrete observations/actions, and thus assume that data streams have been preprocessed, e.g., sensor data have been processed by activity recognisers.

Our GR method aims to overcome several challenges. First, the defined initial world state could be inaccurate; for instance, if an item or agent (e.g., cup or human) is occluded its location is indeterminable, and thus possibly defined incorrectly. Second, the observed agent could act suboptimally [44]; therefore, all plans (including suboptimal plans) are represented within the underlying structure generated by our approach. Third, actions could be missing from the sequence of observations [40], e.g., due to sensor malfunction or occlusions. Finally, an observation should be rapidly processed, so there is little delay in determining the observee's goal. The cited GR works have investigated handling suboptimal observation sequences and handling missing observations, but they do not consider inaccurate initial states.

We define the term inaccurate initial state as an initial state containing fluents (i.e., non-static variables) whose value is unknown (i.e., undefined) and/or incorrect (i.e., set to the wrong value). Inaccurate initial states have been handled by task planners [5, 38]. Moreover, GR with probabilistic, partially observable state knowledge and stochastic action outcomes has previously been investigated [26, 45, 60]; however, these systems require the probability of each state and action outcome to be known (and thus defined within the GR problem). GR with incomplete domain models, i.e., problems containing actions with incomplete preconditions and effects, have also been considered [41] but the initial state was assumed to be accurately represented. Our system makes no assumptions about the correctness of the initial value assigned to a fluent.

In this paper, we aim to answer two research questions. i) Can a structure similar to those created by library-based approaches be generated from a PDDL

defined GR problem? ii) When the initial state is inaccurately defined, how can a goal recognition approach be prevented from suffering a major loss of accuracy?

To answer these questions, we develop a novel technique for transforming GR problems into an Action Graphs, a structure inspired by AND/OR trees. Leaf nodes correspond to actions and are connected to their dependencies via operator nodes. Operator nodes include `DEP` (short for dependencies), `ORDERED-AND`, `UNORDERED-AND` and `OR` nodes. After transforming the action definitions and world model into an Action Graph, the Action Graph's nodes are labelled with their distance from each hypothesis goal, i.e, each goal the observee could be intending to achieve. Both these processes are performed offline. For each observation, the online process updates the goal probabilities based on either the distance the observed action's node is from each goal or the change in distance. Our distance measure is based on the number and type of nodes traversed to reach the node in question from an action node that results in the goal state being reached. The goal(s) with the highest probability are returned as the set of candidate, i.e., predicted, goals. An Action Graph does not contain a perfect representation of all plans; as mentioned by Pereira et al. [40], unlike task planning, this is not a requirement of GR. A conceptual overview of our system is provided in Fig. 1.

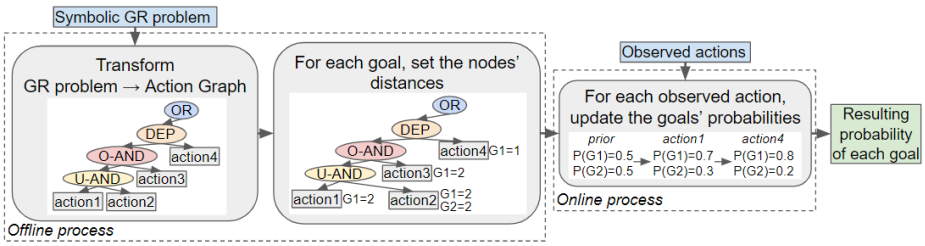


Fig. 1: Conceptual overview of the goal recognition process described in this paper. As indicated by the grey boxes, our approach contains three main processes: (1) create the Action Graph (see Section 4), (2) label the Action Graph's nodes with how many steps away from each goal state they are (see Section 5), and (3) use these labels/distances to update the goals' probabilities when an action is observed (see Section 6).

Our previous work on GR employed an acyclic (rather than cyclic) Action Graph [18], and thus for many domains did not achieve a high accuracy. Moreover, our previous method cannot handle an inaccurate initial state. Action Graphs have also been applied to goal recognition design, in which the aim is to reduce the number of observations required to determine an agent's goal [17]. This paper introduces a novel method for inserting the actions into an Action Graph and presents an alternative approach for updating the goals' probability.

The remainder of this paper is structured as follows. Section 2 presents some background information. A formal definition of our Action Graph structure is provided in Section 3, and Section 4 describes the algorithm that generates the Action Graph. Section 5 introduces our distance measure and how the nodes are labelled with their distance from each goal. The different goal probability update rules, that are executed when an observation is received, are described in Section 6.

Our experimental results, discussed in Section 7, show that our GR method is unaffected by inaccuracies in the initial state.

2 Background

Symbolic task planning and goal recognition problems are often defined in Planning Domain Definition Language (PDDL) [35], a popular domain-independent language for modelling the behaviour of deterministic agents. A PDDL defined problem includes action definitions, objects, predicates and an initial state; an example of each is provided in Listing 1. Our GR approach transforms a PDDL problem into a multi-valued problem by running the converter of [19]. The multi-valued problem is then transformed into an Action Graph. The goal of this section is to provide readers unfamiliar with PDDL and goal recognition the background information required to understand the data provided as input and the notations used. This section first describes why a multi-valued representation is used. The task planning and GR (also known as inverse planning) problem definitions are provided, in the subsections, from a multi-valued problem perspective.

Listing 1: Example of a PDDL defined action, set of objects, set of predicates and initial state. Based on the International Planning Competition's (IPC's) Easy-IPC-Grid domain[1].

```
# Example action definition:
(:action move  :parameters (?1 ?2 - position)
  :precondition (and (at ?1) (not (at ?2)) (adjacent ?1 ?2) )
  :effect (and (at ?2) (not (at ?1)) )
)
# Example set of objects:
(:objects 1_1 1_2 - position)
# Example predicates:
(:predicates (at ?1 ?2 - position) (adjacent ?1 ?2 - position))
# Example initial state:
(:init (at 1_1) (adjacent 1_1 1_2) (adjacent 1_2 1_1) )
```

To create a concise, grounded representation of a problem, a PDDL defined problem is often converted into a multi-valued representation [19, 20]. This representation uses finite variables rather than boolean propositions. For example, rather than a `move(1_1 1_2)` action (which symbolises an agent moving from grid position `1_1` to `1_2`) removing the proposition `(at 1_1)` from the current state and inserting the proposition `(at 1_2)`, a variable, i.e., fluent, that represents the agent's location is changed from `(at 1_1)` to `(at 1_2)`. This enables a more concise representation of the problem to be produced, from which the relations between the different propositions can be extracted. Moreover, a (grounded) action is only created, from an action definition, if its static preconditions appear in the PDDL defined initial world state. For example, to create the `move(1_1 1_2)` action, positions `1_1` and `1_2` must be adjacent. Which locations are adjacent can be statically defined; in other words, no action modifies which locations are adjacent. Further details on the benefits of this representation are given in [20].

[1] `http://www.icaps-conference.org/index.php/Main/Competitions`

2.1 Symbolic task planning

In symbolic task planning, a problem contains a single goal state, and task planners, e.g., Fast Downward [19], find the appropriate set of actions, i.e., a task plan, that can transform the initial world state into the desired goal state. Definitions for states, actions and planning problems are provided below.

Definition 1 *Planning Problem:* A planning problem P can be defined as $P = (F, I, A, G)$, where F is a set of fluents, I is the initial state, G is a goal state, and A is a set of actions [12, 15].

Definition 2 *Fluent:* A fluent ($f \in F$) is a state variable.

When assigned a value, a fluent can be represented by a grounded predicate. Grounded predicates are also called atoms. For instance, `(at 1_2)` is an atom which denotes that the observed agent is at the position `1_2`.

Definition 3 *State:* A state contains all fluents, each of which is assigned a value.

The initial state (I) contains all fluents; whereas, the goal (G) could be a partial state, containing a subset of fluents. To transition between states, the value of fluents are altered by actions. An action is formally defined as follows:

Definition 4 *Action:* An action (a) is comprised of a name, a set of objects, a set of preconditions (a_{pre}) and a set of effects (a_{eff}). Preconditions and effects are composed of a set of valued fluents. Preconditions can contain `or` and `and` statements.

Action a is applicable to state s if the state is consistent with the action's preconditions. Applying action a to state s will result in state s', where $a_{eff} \subseteq s'$ and $\forall(f \in s', f \notin a_{eff}) : (f \in s)$

Definition 5 *Planning Problem Solution:* A solution to a planning problem is a sequence of actions $\pi = (a0, a1, ..., ai \in A)$ such that applying each action in turn starting from state I results in a state (s^i) that is consistent with the (partial) goal state G, i.e., $s^i \supseteq G$.

Planners search for the optimal solution to a planning problem. An optimal solution is the solution with the lowest possible cost. In our work the cost of an action is 1, and thus the cost of a plan is equivalent to its length.

2.2 Goal Recognition

Goal recognition is often viewed as the inverse of planning, as the aim is to label a sequence of observations with the goal the observed agent is attempting to reach. This section provides the formal definition of a GR problem, and describes the observation sequences and output of our GR approach.

Definition 6 *Goal Recognition Problem:* A GR problem is defined as $T = (F, I, A, O, \mathcal{G})$, where $\mathcal{G}$ is the set of all possible (hypothesis) goals and O is a sequence of observations [44].

Definition 7 *Observations:* O is a sequence of observed actions (observations), i.e., $O = (a1, a2, ..., ai \in A)$.

A completed sequence of observations with no missing actions can be applied to an initial state I to reach a goal state $G \in \mathcal{G}$. This sequence can also be incomplete, have missing observations or/and be suboptimal. An incomplete sequence of observations contains the first N actions that are required to reach a goal; in other words, the goal has not yet been reached. An action could be missing from anywhere within a (incomplete or complete) sequence of observations. Observations are suboptimal if any number of additional, unnecessary actions have been performed to reach the goal.

GR approaches attempt to select the real goal from the set of hypothesis goals $\mathcal{G}$. Our GR approach produces a probability distribution over the hypothesis goals, i.e., $\sum_{i=1}^{|\mathcal{G}|} P(G_i|O) = 1$. In other words, we aim to find the likelihood of a given observation sequence O under the assumption that the observee is pursuing a goal G_i, i.e., $P(O|G_i)$. The goal(s) with the highest probability are returned as the set of candidate goals $\mathcal{C}$. As goals can be equally probable, there can be multiple candidate goals, i.e., $|\mathcal{C}| \geq 1$. Nevertheless, we assume that there is only a single real goal. Note, our evaluation metrics (see Section 2.7.1) take into account that multiple goals could be returned.

3 Action Graph Structure and Formal Definitions

Action Graphs model the possible order constraints between actions by linking actions (dependants) to their dependencies. They are constructed of action nodes and operator nodes, namely, `DEP` (short for dependencies), `ORDERED-AND`, `UNORDERED-AND` and `OR` nodes. Action nodes are always leaf nodes and their dependencies are conveyed through their connections (via operator nodes) to other actions. This section defines dependencies, provides a definition of an Action Graph, describes how the Action Graph structure links actions to their dependencies and briefly mentions related structures.

Definition 8 *Action's Dependencies* The set of dependencies of action $a \in \mathcal{A}$ is formally defined as: $D(a) = \{a' \mid (a'_{eff} \cap a_{pre}) \neq \emptyset\}$.

Definition 9 *Action's Dependant* Action a is a dependant of action a' if $a' \in D(a)$.

In other words, action a' is a dependency of action a if at least one effect of a' fulfils at least one of a's preconditions, i.e., $a' \in D(a)$ if $a'_{eff} \cap a_{pre} \neq \emptyset$. In that case, action a is called the dependant of the dependency a'. The order in which dependencies are likely to be observed can be conveyed by the nodes of an Action Graph.

Definition 10 *Action Graph* $AG = (N^O, N^A, E)$, where N^O is a set of operator nodes, N^A are action nodes and E are edges [2]. Operator nodes are of type `DEP`, `UNORDERED-AND`, `OR` and `ORDERED-AND` nodes, i.e., $N^O = (N^{OR}, N^{DEP}, N^{O\text{-}AND}, N^{U\text{-}AND})$. The root node is of type `OR`. All nodes (except the root) have a set of parents. All operator nodes (N^O) have a set of children, those children can be operator nodes or action nodes. N^A are leaf nodes.

The operator node types are described in the list below and depicted in Figure 2. The precedes operator $\prec$ denotes that the list of actions on the left precede (are dependencies of) the action on the right. Standard maths notation is used to denote if a set of actions is unordered or ordered, that is, curly brackets denote the actions are unordered and angle brackets show the actions are ordered [47]. Moreover, rather than writing *or* constraints as two statements, e.g., $a4 \prec a1$ OR $a5 \prec a1$, a shortened form is given, e.g., $or(a4, a5) \prec a1$.

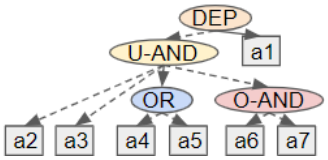


Fig. 2: The different types of order constrains on actions that achieve `a1`'s preconditions, i.e., $\{a2, a3, or(a4, a5), \langle a6, a7\rangle\} \prec a1$. Solid arrows point to the dependant and dashed arrows point to the dependencies. `UNORDERED-AND` is shortened to `U-AND` and `ORDERED-AND` to `O-AND`.

- `DEP` nodes indicate that an action's dependencies are performed before the action itself, e.g., $D(a1) \prec a1$. The second (i.e., last) child of a `DEP` node is the action node itself; the first child could be of any type.
- `UNORDERED-AND` nodes denote that different dependencies set different preconditions (and there are no order constraints on the dependencies), e.g., if $a2 \in D(a1)$, $a3 \in D(a1)$ and $(a1_{pre} \cap a2_{eff}) \neq (a1_{pre} \cap a3_{eff})$ then $(a2 \wedge a3) \prec a1$.
- `OR` nodes express the multiple (alternative) ways a precondition can be reached, e.g., if $a4 \in D(a1)$, $a5 \in D(a1)$ and $(a1_{pre} \cap a4_{eff}) = (a1_{pre} \cap a5_{eff})$ then $or(a4, a5) \prec a1$.
- `ORDERED-AND` nodes indicate there are order constraints between an action's dependencies. Such constraints are required when executing one dependency could unset the preconditions of another. For example, if $a6 \in D(a1)$, $a7 \in D(a1)$ and both $a6_{pre}$ and $a7_{eff}$ contain the same fluent but with different values, then $a6$ is performed before $a7$, i.e., $\langle a6, a7\rangle \prec a1$. This is because the effects (fluents) of $a7$ are preconditions of $a1$ but to perform $a6$ those fluents must be assigned a different value. If these constraints are cyclic, e.g., $\{\langle a6, a7\rangle, \langle a7, a6\rangle\} \prec a1$, then the constraint is ignored; in other words, the dependencies are considered to be unordered.

Dependencies are actions, and thus they can also have dependencies. For example $a8$ could depend on $a9$, which depends on $a10$, i.e., $a10 \prec a9 \prec a8$. If an

[2] N^x denotes a set of nodes that are of a specific type (x).

action has dependencies, its only parent is the `DEP` node linking it to its dependencies (and dependants). Thus, continuing with the example, the left child of $a8$'s parent `DEP` node is $a9$'s parent `DEP` node.

Cyclic dependencies can also occur, e.g., $a1$ could depend on $a2$ which depends on $a1$ (i.e., $...a2 \prec a1 \prec a2...$). This causes cycles to appear within the Action Graph. These cycles can also be caused by indirect dependencies, e.g., $...a2 \prec a3 \prec a1 \prec a2...$ in which $a1$ is a dependency of $a2$ and $a2$ is an indirect dependency of $a1$.

Our Action Graph structure does not contain states. An Action Graph only captures information about which actions fulfil each action's preconditions. This is similar to the structures of library-based intention recognition and planning approaches [21, 28, 49, 52]. For instance, Goal-Plan trees contain (sub)goals with plans that can contain subgoals [49, 52]. Goal-Plan trees do not contain knowledge of the environment's current state. In particular, the Action Graph structure was inspired by the work of Holtzen et al. [21], who represented a library of plans as AND/OR trees. Differently, our approach takes PDDL rather than a library of plans as input, and enables suboptimal and cyclic dependencies to be represented.

Moreover, the definition of a dependency is similar to causal links from Partial-Order Causal Link (POCL) planning [16]. Like dependencies, a causal link expresses that an action's preconditions are contained within another action's effects. Differently, POCL structures represent complete plans (to reach a single goal state from the initial state), edges rather than nodes are used to denote the order constraints and they can contain ungrounded actions. As GR does not require a completely valid plan, Action Graphs are simpler to construct than POCL structures.

4 Cyclic Action Graph creation

Our goal recognition method creates an Action Graph, labels the nodes with their distance from each goal, then for each observation updates the goals' probability. This section describes how an Action Graph is generated from a GR problem. The modifications to the preprocessing step, that transforms a PDDL problem into a multi-valued problem, are described. Subsequently, the action insertion algorithm is detailed, followed by an example.

4.1 Preprocessing: multi-valued problem generation

This paper only provides the details of the transformation, from a PDDL defined problem to a multi-valued problem, that are key to understanding our approach and that differ from [19]. A single goal statement is required by the coverter of Helmert [19]; therefore, prior to calling the converter, a goal statement is created by placing all hypothesis goals ($\mathcal{G}$) into an `or` statement, i.e., $G = or(G_1, G_2, ..., G_{|\mathcal{G}|} \in \mathcal{G})$.

The converter's parameter, to keep all unreachable states, is set to true and, after parsing the PDDL, all groundings of the actions' effects are inserted into the initial state (I). This forces actions, and all fluents' values, to be inserted into the resulting representation even if the actions' fluent preconditions, and thus

possibly the goals, are unreachable from the defined (original) initial state. For instance, if an agent's location is missing from I, e.g., because it is unknown, and no transition between an unknown and known location exists, then `move` actions would not be inserted as their preconditions can never be met. To prevent this, all `(at ?location)` groundings are inserted into I. Additional static atoms are not inserted into I; thus, continuing with our example, `move(1_1 1_2)` is only appended to the set of actions A if `(adjacent 1_1 1_2)` is declared in the defined initial state.

4.2 Inserting actions into an Action Graph

An Action Graph is initialised with an `OR` node as the root; then each action ($a \in A$) is inserted into the graph in turn by connecting it to its dependencies. Actions can be inserted in any order. Finally, the graph is adjusted so only the Goal Actions' parent `DEP` nodes are connected to the root. This process is detailed below and the pseudo-code is provided in Appendix A.

If an action has no dependencies, because either there are no actions that fulfil its preconditions or it has no preconditions, it is simply appended to the root's children. In all other cases, the root is linked to a new `DEP` node. The `DEP` node's two children are set to an `UNORDERED-AND` node, proceeded by the action node itself. If this action node was already created, because it is a dependency of an already processed action, the action node's prior parents are moved to be the `DEP` node's parents.

The `UNORDERED-AND` node's children are set to one or more of the following: the action nodes (or parents) of the dependencies, `OR` nodes if there are multiple ways in which a precondition can be met, and/or `ORDERED-AND` nodes. `OR` nodes' are inserted by setting their children to the action nodes of the dependencies that set the same precondition(s). If a dependency has dependencies, the corresponding child becomes the dependency's parent. This is because actions that have dependencies can only ever have a single parent, of type `DEP`. Note: if an operator would only have one child, the operator node is not inserted.

`ORDERED-AND` nodes indicate that there are order constraints on the dependencies themselves. This is detected by checking if a fluent has a value in a dependency's preconditions which is different in another dependency's effects (see Section 3); and thus the former dependency must be performed first. If this constraint is bidirectional/cyclic, the `ORDERED-AND` node is not inserted; instead, the dependencies become the children of the `UNORDERED-AND` node. Only the preconditions/effects of direct dependencies are checked, the algorithm does not check if a dependence's dependency could undo/unset a dependency's precondition. Performing this check would be computationally complex and a perfect representation of the plans to reach of the actions' effects is not required.

The `ORDERED-AND` node's children could also be of type `UNORDERED-AND` or `OR`. When multiple dependencies could unset a dependency's preconditions, an `UNORDERED-AND` is inserted as the child of the `ORDERED-AND` node's right-branch. Moreover, the dependencies that set the same precondition(s) of the dependant are grouped together as the children of an `OR` node. Therefore, if one of these dependencies is affected by (or effects) another dependency, the `OR` node becomes the `ORDERED-AND`

(or UNORDERED-AND) node's corresponding child. Without this feature, the graph's structure would become more complex, i.e., be of greater depth and/or breadth.

4.3 Identifying Goal Actions

An action is a Goal Action if its effects fulfil a goal's atoms, i.e., $a_{eff} \supseteq G$, where $G \in \mathcal{G}$. After all actions have been inserted, the root node's children are modified so that only the Goal Actions are attached to the root. If multiple actions are required to fulfil a goal, e.g., $(a1_{eff} \cup a2_{eff}) \supseteq G$, then an auxiliary Goal Action (a^x) is created. Auxiliary Goal Actions are linked to the multiple actions that fulfil the goal via a DEP node, e.g., $\{a1, a2\} \prec a^x$. They are connected to their dependencies, i.e., the goal's dependencies, in the same way as all other actions are.

Identifying and creating Goal Actions simplifies traversing the graph to find all nodes belonging to a single goal. All children, including indirect children, of a Goal Action's parent DEP node could appear in a plan (from any initial state) to reach the goal the Goal Action fulfils. Therefore, the graph can be traversed, in a depth-first or breadth-first manner, to find all the nodes, and thus actions, belonging to a goal.

4.4 Example

An example is provided, in this section, to demonstrate how our creation algorithm works for the grid-based navigation problem depicted in Fig. 3. Fig. 4 shows the Action Graph after each action, and its dependencies, have been inserted. The four insertions, detailed below, were selected to show the different structural features of an Action Graph. A figure with all actions inserted into the graph would be unreadable, and thus is not provided.

The example starts by inserting the goal action, `move(2_0 1_0)`. The preconditions of `move(2_0 1_0)` are met by executing one of two possible actions, i.e., or(`move(1_0 2_0)`,`move(2_1 2_0)`) $\prec$ `move(2_0 1_0)`; therefore, it is inserted by connecting it to its dependencies via a DEP node and a OR node (see Fig. 4a). Likewise, when `move(1_0 0_0)`, whose preconditions are reached by one of three

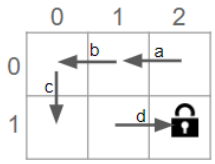
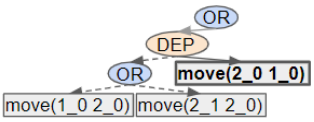


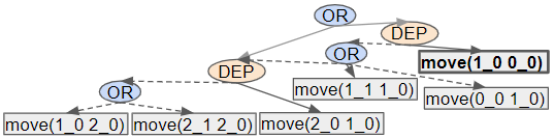
Fig. 3: Example problem from the Easy-IPC-Grid domain. Before an agent can move to position (2,1), it must be unlocked. The lettered arrows indicate the actions inserted to produce the sub-figures of Fig. 4. Based on the GR Easy-IPC-Grid problems developed by Ramírez and Geffner [43], based on a domain from the official IPC[1].

actions (i.e., or(`move(2_0 1_0)`, `move(1_1 1_0)`, `move(0_0 1_0)`) $\prec$ `move(1_0 0_0)`), is inserted, an `OR` node is created. As one of its dependencies has already been inserted, the appropriate child of the `OR` node is set to `move(2_0 1_0)`'s parent `DEP` node. This is shown in Fig. 4b.

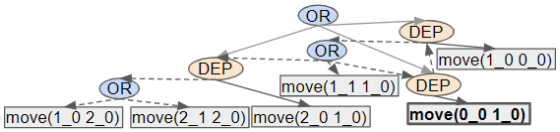
Inserting `move(0_0 1_0)` causes the graph to become cyclic (Fig. 4c) because it depends on one of its dependants, i.e., `move(1_0 0_0)` $\prec$ `move(0_0 1_0)`. Fig. 4d displays the graph after `move(1_1 2_1)` has been inserted. This action requires location `2_1` to be unlocked with `key1`, and thus its dependencies include `unlock` actions. As `unlock` actions' preconditions contain the location of the agent, they must be performed prior to the move actions required by the dependant. Therefore, an `ORDERED-AND` node is created during the insertion of `move(1_1 2_1)`, i.e., $\langle or$(`unlock(2_1 2_0 key1)`, `unlock(2_1 1_1 key1)`), or(`move(0_1 1_1)`, `move(2_1 1_1)`, `move(1_0 1_1)`)$\rangle \prec$ `move(1_1 2_1)`.



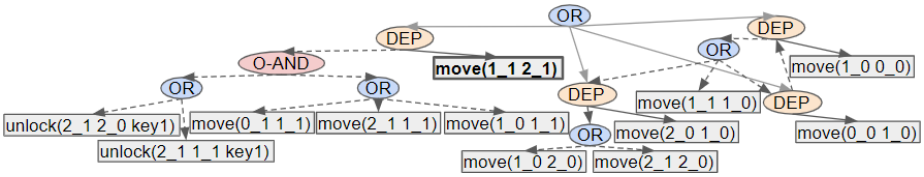
(a) The Action Graph after `move(2_0 1_0)` and its dependencies have been inserted.



(b) `move(1_0 0_0)` has been inserted into the Action Graph along with its dependencies. As `move(2_0 1_0)` was previously inserted, the new `OR` node was connected to it via its parent (`DEP`) node.



(c) `move(0_0 1_0)`'s dependency was already inserted into the graph. Connecting it to its dependency has caused a cyclic to occur.



(d) The order constrains on `move(1_1 2_1)`'s dependencies required an `ORDERED-AND` node to be inserted as performing a move action could undo a precondition of an action that unlocks position `2_1`.

Fig. 4: Example of the steps taken to insert four actions, and their dependencies, into an Action Graph. This example has been simplified, i.e., in the original Easy-IPC-Grid problems, keys have different shapes. Solid pale arrows show the root node's connections (prior to the Goal Actions being discovered).

5 Node distance initialisation

Each node has a set of distances associated with it, which indicate how far the node is from each goal, i.e., the number of `DEP` and `ORDERED-AND` nodes that must be traversed to get from the Goal Action's parent to the node in question. These distances are set by means of a breadth-first traversal (BFT). A BFT was implement because this will result in the nodes being visited in the order they are likely to be performed. An explanation of this algorithm is provided, followed by an example. The pseudo-code can be found in Appendix B.

5.1 Node value initialisation algorithm

During the BFTs, that start from each Goal Action's parent node, the current node's distance is set, the count (i.e., distance measure) is increased if the node is of type `DEP` or `ORDERED-AND`, and each of the node's children are pushed onto the BFT-queue. This distance measure provides an indication of how far each node is from each goal whilst attempting to minimise favouring shorter plans (see Section 6 for the calculations of the goal probabilities). The same node could be visited multiple times during a BFT; however, if the current distance/count is greater than or equal to the node's already assigned distance, it is not reprocessed. As well as allowing the shortest distance to be assigned to each node, this prevents an endless loop from occurring when two actions depend on each other (e.g., $...a2 \prec a1 \prec a2...$).

As an action could appear in a plan multiple times, some nodes require multiple distances for the same goal; this is the case for the descendants of `ORDERED-AND` nodes' right branch. Therefore, a node contains a map for each goal, from the last traversed `ORDERED-AND` to the node's distance from the goal via the `ORDERED-AND` node. When the right branch of the `ORDERED-AND` node has been fully observed, the distance of the node, returned when calling a get distance method, will be the distance associated with that `ORDERED-AND` node. As the initial state is unknown and plans are not perfectly represented, the distances assigned to the left branch of `ORDERED-AND` nodes are not based on the depth of the right branch.

Labelling nodes with multiple distances per goal increases the worst case time complexity from $O(n^2)$ to $O(n^3)$, with respect to the number of actions. This is because each action in the graph could be a dependency of all other actions; thus, for all actions all other actions could be visited. When labelling the nodes with multiple values this process could be repeated n times. Therefore, to help minimise the number of nodes the BFTs traverse, when an `UNORDERED-AND` node is reached, its children's (including indirect children's) distance is not associated with the prior `ORDERED-AND` node(s). Developing this component greatly reduced (≈halved) the run time of our experiments (Section 7.3.2) and had negligible impact on the accuracy of our approach.

The offline component of our system finishes by setting the prior probability of each goal. We chose to use a uniform prior probability as, since no actions have been observed, all goals are assumed to be equally likely.

5.2 Example

The Action Graph depicted in Fig. 5 shows the resulting action nodes' distance from each goal for a simplified version of the Kitchen domain by Ramírez and Geffner [43]. In this example, there are two Goal Action nodes, namely, `pack-lunch()` and `make-dinner()`. By executing the BFTs described above, each node is labelled with their distance from each goal. This example will be used in Section 6, to demonstrate how an observation affects the goals' probability, and thus why this node value initialisation procedure has been implemented.

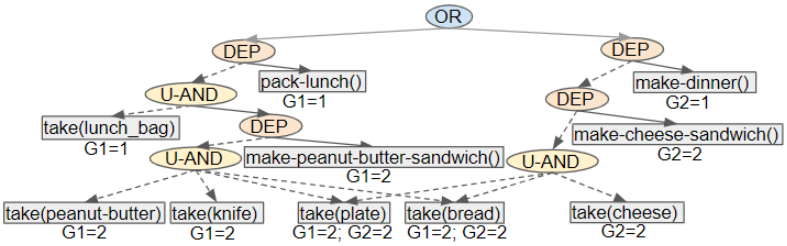


Fig. 5: Example of an Action Graph with the action nodes labelled with their distance from each goal. `G1` represents the goal `(made_lunch)`, which is reached by performing the `pack-lunch()` action and `G2` represents `(made_dinner)`, which is reached by performing the `make-dinner()` action. This example is based on the Kitchen domain developed by Ramírez and Geffner Ramírez and Geffner [44] based on the work of Wu et al. [57]. To make this figure readable, it has been simplified, i.e., many nodes and edges are not included.

6 Updating the goal probabilities

When an action is observed, the probability associated with each goal is updated based on either its distance from the observed action or the difference between its distance from the prior observation and the current observation. These two update rules are described in turn along with their advantages and disadvantages. The experiments section presents results for both these update rules separately, as well as combined. The pseudo-code, for the rules combined, is provided in Algorithm 1.

6.1 Update rule 1: Distance from observed action

Each goal's probability is updated based on how close the goal is to the observed action and how unique the observation is to the goal. The probabilities of the goals closest to the observation are increased, whilst those furthest from the observation are decreased. If an observation only belongs to a single goal, that goal's probability is increased and all other probabilities are decreased. This is performed by multiplying each goal's probability by its distance from the observed action's node divided by the sum of all goals' distances (lines 7-10); then normalising the

Algorithm 1: Update the goal probabilities.

Data: o^t observed action, o^{t-1} previously observed action, $\mathcal{G}$ set of hypothesis goals with current probability
Result: $\mathcal{G}$ set of hypothesis goals with updated probability

1 **if** $o^{t-1} \neq null$ ***and*** *areConnectedViap/OAndNodes*(o^t, o^{t-1}) **then**
2 **foreach** $G \in \{G' \mid G' \in \mathcal{G}, o^t.hasDisFromGoal(G')\}$ **do**
3 $c(G) = \sigma(o^{t-1}.\text{getDisFromGoal}(G) - o^t.\text{getDisFromGoal}(G))$
4 $v(G) = P(G)(1 + c(G))$
5 **end**
6 **else**
7 **foreach** $G \in \mathcal{G}$ **do**
8 $c(G) = \frac{o^t.\text{getDisFromGoal}(G)}{\sum_{G' \in \mathcal{G}} o^t.\text{getDisFromGoal}(G')}$
9 $v(G) = P(G)(1 + c(G))$
10 **end**
11 **end**
12 **foreach** $G \in \mathcal{G}$ **do** $P(G) = \frac{v(G)}{\sum_{G' \in \mathcal{G}} v(G')}$ ▷ `probabilities sum to 1`
13 updateNodeDistancesIfo-andLeftBranchFullyObserved(o^t)

resulting values (line 12). Note, if the observation is not within a plan to reach the goal G, 0 is returned by the *getDisFromGoal* method (line 8) so that $c(G) = 0$ and, so long as another goal's plan contains the action, its probability is reduced.

For the example shown in Fig. 5, there are two goals, both with a prior probability of 0.5. When the `take(plate)` action is observed, the resulting probabilities are unaltered as its node's distance to each goal is equal. More nodes must be traversed to reach `take(plate)` from `pack-lunch()`, than from `make-dinner()`. Nevertheless, the goal with a shorter plan was not favoured as the distance counter (see Section 5) was only increased when a `DEP` or `ORDERED-AND` node was traversed. If `take(knife)` is observed, the probability of making a pack lunch is increased, i.e., $P($`(made-lunch)`$) = 0.67$ and $P($`(make-dinner)`$) = 0.33$, as the observed action is unique to this goal.

The main disadvantage of this approach is that the probabilities of goals with shorter, strongly ordered, plans are increased more than those with longer plans. Therefore, the list of returned candidate goals $\mathcal{C}$ often contains the goal(s) with a shorter plan. For instance, if an incomplete sequence of observations contains actions that approach both G1 and G2, whichever of these two goals has the shortest plan length will be returned as a candidate goal, the other will not be. The subsequent update rule aims towards mitigating this disadvantage.

6.2 Update rule 2: Change in distance from the observed actions

If the previous observation (o^{t-1}) and the current observation (o^t) are connected via a `DEP` or `ORDERED-AND` node, the goal probabilities are updated based on the change in distance, i.e., the difference between the goal's distance from the previous and current observations (lines 2-5 of Algorithm 1). To check if the observations are connected, an upwards traversal (in a depth-first manner) is performed, starting from the action node of o^{t-1}, to find a `DEP` or `ORDERED-AND` node whose right branch's child is the action node of o^t.

If the list of observations is not missing any actions, the change in distance will always be 1, 0 or -1. As an observation could be missed (e.g., due to sensor failure), our algorithm needs to account for the difference being within a wider range of values. A negative difference indicates the observee moved further from the goal, whereas a positive difference indicates they moved closer. The sigmoid function converts the difference into a value between 0 and 1 (i.e., σ from line 3); the goal's value is multiplied by this (line 4) then normalised (line 12). If either observation does not belong to the goal, the value of $v(G)$ is equivalent to setting the result of the sigmoid function to 0; in other words, the difference is $-\infty$. This update rule results in the probability of the goal the observee is moving towards at the highest rate to be increased the most.

When this rule is used independently from the first rule, if the previous observation is null or the current and previous observations are not connected via a `DEP` or `ORDERED-AND` node (as detailed above), then $c(G) = 0.5$ for the goals dependent (or indirectly dependent) on the current observation and $c(G) = 0$ for all other goals. This prevents the goals that have shorter plans from being favoured; however, goals for which the observation appears in a (very) suboptimal plan are treated equally to those for which an optimal plan contains the observation.

In the example depicted in Fig. 6, if the observee moves from position `2_1` to `1_1` then to `0_1`, the goal probabilities remain equal. As the distance to both goals is reduced at the same rate, the real goal is indiscernible. If the observee were to move vertically, and thus step towards one goal (and away from the other), the corresponding goal's probability is increased, e.g., observing `move(2_1 1_1)` then `move(1_1 1_0)` results in $P(G1) = 0.58$ and $P(G2) = 0.42$.

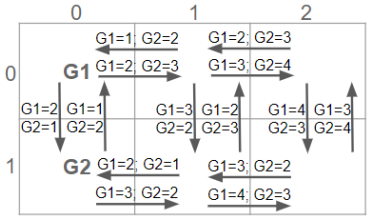


Fig. 6: Action nodes' distance from each goal, represented on a depiction of a grid-based navigation environment. `G1` and `G2` are goals and arrows represent move actions.

6.3 Processes common to update rules 1 and 2

In both update rules, $1 + c(G)$ is calculated, rather than just $c(G)$, so that a goal's probability is never set to 0. If the probability of a goal were to be set to 0, it cannot be increased; thus, the heuristic would not be able to recover from receiving an incorrect (noisy) observation. These update rules, along with the graph's structure, enable our system to handle noisy observations as well as suboptimal plans and missing observations.

When an action has been observed, which operator nodes have been completed are updated by traversing up the graph, in a depth-first manner, from the observed action's node (line 13). `OR` nodes are set as completed if one of their children has been completed, `DEP` nodes are complete if the child connected to their right branch has been observed and `UNORDERED-AND` nodes are set as completed if all their children are. If a node is not set to completed, its parents are not traversed. When an `ORDERED-AND` node's left branch has been completed, the nodes attached to its right branch are informed, so that their distance associated with that `ORDERED-AND` node is used when the next observation is received (as described in Section 5).

7 Experiments

Through experiments we aim to demonstrate the accuracy of our GR approach, after 10, 30, 50, 70 and 100 % of actions in a plan have been observed, on 15 different domains. This section describes the evaluation metrics, followed by the setup and results of the different experiments. A comparison between our different update rules and the goal completion heuristic, namely, h_{gc}, by Pereira et al. [39, 40] is provided, on problems for which differing percentages of fluents have been set to incorrect values. Our method is then compared to h_{gc} on a dataset containing GR problems with a known, and thus correctly defined, initial world state.

Pereira et al. [39, 40] recently improved the accuracy and computational time of GR by finding landmarks, i.e., states that must be reached to achieve a particular goal. After processing the observations, the resulting value of each goal ($G \in \mathcal{G}$) is based on the percentage of its landmarks that have been completed. h_{gc} takes a threshold value as a parameter. Any goals whose value is greater than or equal to the most likely goal's value minus the threshold are included in $\mathcal{C}$. When the threshold is 0.0, like our approach, only the most likely goal(s) are included in $\mathcal{C}$. Therefore, we present the results of their approach for 0.0 as the threshold. The compiled version of h_{gc}, provided by Pereira et al. [39], was ran during the experiments[3]. We provide a detailed comparison to h_{gc} as it has been very recently developed and was shown to outperform alternative methods; other approaches to GR will be discussed in the related work section (Section 8).

The dataset created by Ramírez and Geffner [44] and Pereira et al. [39][4] forms the basis for our experiments. Details on generating the lists of observations and the inaccurate initial states are provided in the setup sections, specific to each experiment. A brief description of each domain is supplied in Appendix C. Experiments were ran on a server with 16GB of RAM and a Intel Xeon 3.10GHz processor.

7.1 Evaluation metrics

Our approach is evaluated on the number of returned candidate goals (i.e., $|\mathcal{C}|$) and standard classification metrics, namely, accuracy (sometimes referred to as

[3] `https://github.com/ramonpereira/Landmark-Based-GoalRecognition/blob/master/goalrecognizer1.1.jar`

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

quality), precision, recall and F1-Score [10, 24]. A definition for each of these is provided below. Subsequently, performance profiles, which are provided to show a comparison of the approaches' run-times, are introduced.

7.1.1 Classification metrics applied to goal recognition

Accuracy ($Q_{P,S}$), precision ($M_{P,S}$), recall ($R_{P,S}$) and F1-Score ($F1_{P,S}$) are provided in Eqs. 1, 2, 3 and 4 respectively. The definitions of TP, FP, FN and TN are provided, from a GR perspective, in Table 1. In these definitions G_P is the actual goal (ground truth) for problem P, $\mathcal{C}_{P,S}$ is the set of candidate goals returned by solution/approach S and $\mathcal{G}_P$ is the set of hypothesis goals. TP is 1 if the true goal is in the set of candidates or 0 if it is not; FN is the inverse of TP; FP is the number of returned candidates that are not the real goal, and TN is the number of goals correctly identified as not the real goal. For each metric, the average over all problems per domain is displayed in the results.

Table 1: Definitions of True Positive (TP), False Positive (FP), False Negative (FN) and True Negative (TN) results of solution/approach (S) on a goal recognition problem (P).

$TP = \begin{cases} 1, & \text{if } G_P \in \mathcal{C}_{P,S} \\ 0, & \text{otherwise} \end{cases}$	$FP = \begin{cases} \lvert\mathcal{C}_{P,S}\rvert - 1, & \text{if } G_P \in \mathcal{C}_{P,S} \\ \lvert\mathcal{C}_{P,S}\rvert, & \text{otherwise} \end{cases}$
$FN = \begin{cases} 0, & \text{if } G_P \in \mathcal{C}_{P,S} \\ 1, & \text{otherwise} \end{cases}$	$TN = (\lvert\mathcal{G}_P\rvert - 1) - FP$

$$Q_{P,S} = \frac{TP + TN}{TP + TN + FN + TP} \tag{1}$$

$$M_{P,S} = \frac{TP}{TP + FP} \tag{2}$$

$$R_{P,S} = \frac{TP}{TP + FN} \tag{3}$$

$$F1_{P,S} = \begin{cases} 2 * \frac{M_{P,S} * R_{P,S}}{M_{P,S} + R_{P,S}}, & \text{if } G_P \in \mathcal{C}_{\mathcal{P},\mathcal{S}} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

Prior goal recognition papers [7, 39, 44] defined the accuracy/quality as the number of times the actual goal appeared in the set of candidate goals, i.e, they did not take $|\mathcal{C}|$ into consideration when calculating accuracy. This resulted in approaches being reported as 100 % accurate, even when more than one candidate goal was returned. In our paper, this is equivalent to recall ($R_{P,S}$). By using the definitions provided in this paper, an approach can only have an accuracy of 1 (i.e., 100 %) if it always returns one candidate goal, i.e., the real goal.

7.1.2 Performance profiles

The computation times (T) are presented in performance profiles, as suggested by Dolan and Moré [6]. This enables the results to be presented in a more readable format, and all datasets can be grouped into a single result to prevent a small number of problems from dominating the discussion. To produce the performance profile an approach ($S \in \mathcal{S}$), i.e., of our Action Graph approach and of h_{gc}, the ratio between its run-time ($T_{P,S}$) and the quickest run-time for a problem ($P \in \mathcal{P}$) is calculated, as shown in Eq. 5. Eq. 6 calculates the percentage of problems an approach solved when the ratio is less than a given threshold, τ. When $\tau = 0$, the resulting $P_S(\tau)$ of an approach is the percentage of problems it solved quicker than the other approach. How much τ must be increased for 100% of problems to be solved depends on how far off the best approach that approach is.

$$\Gamma^T_{P,S} = \frac{T_{P,S}}{\min\left(T_{P,S} : S \in \mathcal{S}\right)} \tag{5}$$

$$P_S(\tau) = \frac{1}{|\mathcal{P}|}|P \in \mathcal{P} : \Gamma^T_{P,S} \leq \tau| \tag{6}$$

7.2 Goal recognition with an inaccurate initial state

The main aim of our approach is to be able to perform GR when the initial state of the environment is defined inaccurately. A fluent's value could be incorrect if it is unknown, and thus incorrectly guessed, or an error has been made while determining the environment's state. Therefore, a dataset containing differing percentages, i.e, 10, 20, 40, 60, 80 and 100 %, of fluents set to incorrect values was produced. How this dataset was generated is discussed, followed by the results produced by our Action Graph approach and h_{gc}.

7.2.1 Setup

A dataset containing problems with varying amounts of fluents set to incorrect values was generated from the dataset containing the first N % of observations, i.e., that was used in the experiments of Section 7.3.3. For each problem, contained in the aforementioned dataset, 10, 20, 40, 60, 80 and 100 % of fluents were chosen at random and their value set to a randomly selected incorrect value. As there are elements of randomness, for each percentage of fluents, 5 problems were created. The changes that can be made to the initial state I were (manually) defined based on the actions' effects.

For instance, in a Zeno-Travel problem, containing 5 cities, 4 people, 3 aircraft and 2 fuel-levels, there are 10 fluents whose initial value can be altered. Each person can be at a city or in an aircraft; thus, a fluent indicating a person's location can be changed to one of the 7 alternative (incorrect) values. Each aircraft has two fluents associated with it, i.e., is in a city and has a fuel-level; both of these can be changed to an alternative value.

These state changes could cause some (or all) goals to be unreachable from the defined initial state (e.g., in the Sokoban domain, the robot could be unable to navigate to a location from which one of the boxes can be pushed) and the

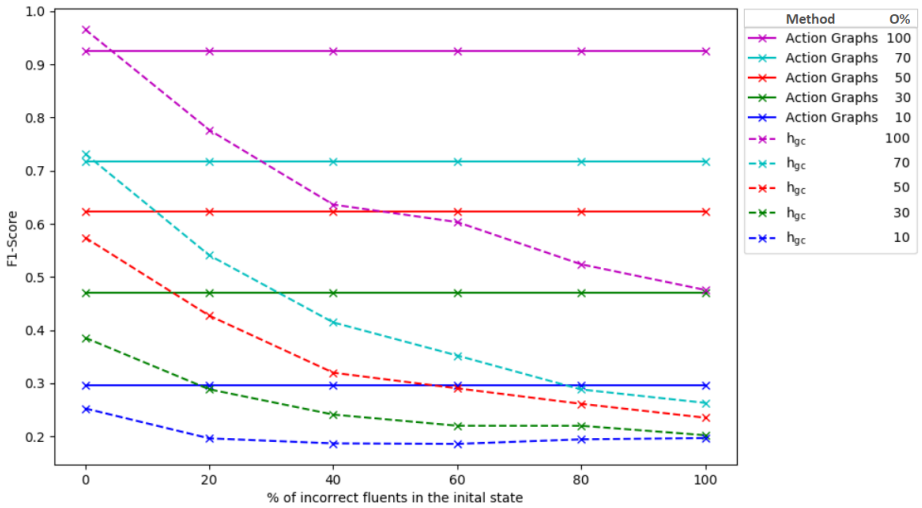


Fig. 7: Graph showing the effect increasing the amount of incorrect fluents in the initial state had on the accuracy of our Action Graph approach and h_{gc} by Pereira et al. [39]. Our Action Graph approach is indicated by solid lines, the dash lines show the approach of Pereira et al. [39]. Each line colour indicates a different % of observations.

initial state itself could be invalid/contradictory (e.g., in the Blocks-World domain, `blockA`'s fluent could express the block is on the table and the gripper's fluent could indicate it is holding `blockA`). The changes made to the initial state are outlined in Appendix C and a detailed table of possible changes can be found at `https://doi.org/10.5281/zenodo.3621275`. No modifications were made to the lists of observations.

7.2.2 Results discussion

The accuracy of our approach was not affected by setting fluents in the initial state to incorrect values, whereas the accuracy of h_{gc} greatly reduced (see Fig. 7 and Appendix D). When building the Action Graph, the initial values of fluents are ignored. As a result, no matter what the initial values of fluents are defined as being, the same Action Graph is produced. The initialising of the nodes' values uses the goal states, but not the initial state. Therefore, the output of our goal recognition approach is unaffected by incorrectly initialised fluents. Previously, researchers have used the initial state in their approach, and as a result, their accuracy will deteriorate. When 20 % of the fluents' were set to incorrect values, a large decrease in the accuracy of h_{gc} was observed, and as this percentage was increased, the accuracy further reduced. For several domains, i.e., Kitchen, Rovers and Intrusion-Detection, the resulting M and R of h_{gc} rose when 100 % of fluents (rather than 80 %) were incorrect. This is because at 100 %, for these domains, all goals were contained in the set of candidate goals, and thus the real goal was contained within $\mathcal{C}$. Other approaches to GR are also unable to handle inaccurate initial states because they attempt to find the plans/states that reach each goal

from the defined initial world state. These are discussed further in the related work section.

7.3 Goal recognition with a known initial state

After describing the experiment setup, this section compares the computational time of our Action Graph approach to h_{gc}. The accuracy of these approaches, when ran on problems with accurate initial states, are then discussed. This includes GR problems where the observations are the first N % of actions in a plan and ones for which the observations are a random N % of actions in a plan (i.e., is missing observations).

7.3.1 Setup

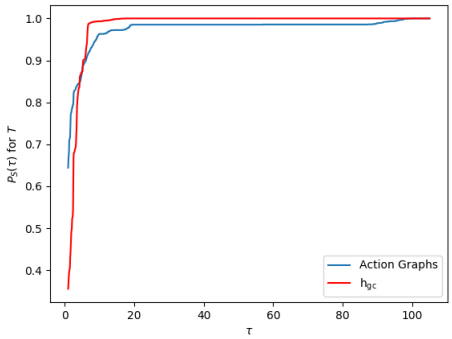
The GR problems in the original dataset[4] contain 10/30/50/70/100 % of observable actions in the plan to reach a goal; these observations/actions were selected at random. Therefore, for each of the original problems that contain 100 % of the observations, we generated GR problems by selecting the first 10, 30, 50, 70 and 100 % of observations. As a task planner (which is not guaranteed to find an optimal plan) was ran to create the problems produced by Pereira et al. [39], some observation sequences are suboptimal.

The accuracy results for our two goal probability update rules (described in Section 6), when ran independently and combined, are presented. In the results table these are named AG1 (i.e., the first update rule), AG2 (the second update rule) and AG3, which is the combination of the two rules (i.e., Algorithm 1).

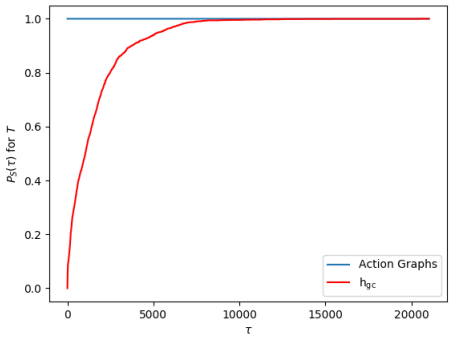
7.3.2 Run-times

The Action Graph heuristic took an average of 0.02 s to process all observations, whereas h_{gc} took 0.66 s. Labelling the nodes with their distance from the goals is computationally expensive; therefore, when the offline processing times (which includes the PDDL to Action Graph transformation steps) are included, Actions Graphs took an average of 2.38 s per problem. The performance profiles are displayed in Fig. 8 and the results per domain are shown in Table 2. These run-times were produced while processing the GR dataset containing the first N % of observations, which contains 2705 problems.

When the whole process is included in the run-times, our approach GR out performed h_{gc} on 64 % of problems; however, the difference in run-time was greater for the problems our solution was slower at (than for the problems h_{gc} was slower on). This is indicated by how much τ must be increased before 100 % of problems were solved. At $\tau = 17.45$, h_{gc} solved all problems and at $\tau = 100.00$ all problems were solved by our approach. If only the online process is included, our approach solves 100 % of problems quicker than h_{gc}, and τ must reach 20426 before h_{gc} solves 100 %. Note: for h_{gc}, the landmarks could be discovered offline, and thus the online computational time reduced. This is only possible if the initial value of each fluent is known in advance.



(a) Run-time comparison, which includes the graph initialisation (offline) time for our approach.



(b) Run-time comparison, which excludes the graph initialisation time for our approach.

Fig. 8: Performance profiles comparing the recognition time of our Action Graph approach to the goal completion heuristic by Pereira et al. [39]. These values were produced using the dataset containing GR problems with the first N % of actions in a plan as observations.

Table 2: The run-times, in seconds, of our Action Graph approach, including and excluding the Action Graph initialisation (i.e., creation and node labelling) time, and h_{gc} [39] per domain. *ALL* is the total/average over all problems.

		Action Graphs (incl. graph initialisation)			Action Graphs (excl. graph initialisation)			h_{gc}		
Domain	$\|probs\|$	$\sum t$	$\bar{t}$	$\pm std$	$\sum t$	$\bar{t}$	$\pm std$	$\sum t$	$\bar{t}$	$\pm std$
Blocks-World	460	109.54	0.24	0.38	0.27	0.00	0.00	213.73	0.46	0.37
Campus	75	5.02	0.07	0.00	0.01	0.00	0.00	22.50	0.30	0.07
Depots	140	193.62	1.38	1.99	0.15	0.00	0.00	108.78	0.78	0.24
Driverlog	140	1964.08	14.03	17.56	0.24	0.00	0.00	96.03	0.69	0.49
DWR	140	243.17	1.74	2.49	0.11	0.00	0.00	116.04	0.83	0.26
Easy-IPC-Grid	305	1269.91	4.16	5.53	16.64	0.05	0.12	174.89	0.57	0.24
Ferry	140	124.39	0.89	1.48	2.23	0.02	0.03	53.17	0.38	0.15
Intrusion-Detection	225	12.64	0.06	0.00	0.04	0.00	0.00	85.84	0.38	0.10
Kitchen	75	3.79	0.05	0.00	0.00	0.00	0.00	20.73	0.28	0.07
Logistics	305	709.73	2.33	5.21	6.39	0.02	0.06	259.74	0.85	0.85
Miconic	140	520.12	3.72	5.75	17.71	0.13	0.25	120.04	0.86	0.66
Rovers	140	207.68	1.48	1.04	0.13	0.00	0.00	119.06	0.85	0.48
Satellite	140	108.35	0.77	0.84	0.20	0.00	0.00	133.58	0.95	0.69
Sokoban	140	92.30	0.66	0.42	0.07	0.00	0.00	125.74	0.9	0.32
Zeno-Travel	140	877.03	6.26	4.68	0.96	0.01	0.01	141.02	1.01	0.43
ALL	2705	6441.37	2.38	6.03	45.15	0.02	0.08	1790.89	0.66	0.51

Both the size and the structure of an Action Graph impact the run-times of our approach. Domains with relatively few actions have much shorter initialisation time (e.g., Kitchen, Intrusion-Detection and Campus). For larger domains, the structure of the graph had a greater impact as the run-time was affected the number of times each node was visited. Our node labelling algorithm, which performs

Table 3: The average size of the Action Graph per domain. *ALL* is the average over all problems. The Nodes column provides the total number of action, `DEP`, `ORDERED-AND`, `UNORDERED-AND` and `OR` nodes.

Domain	Edges	Nodes	Actions	DEP	O-AND	U-AND	OR
Blocks-World	13994.16	1096.39	180.89	180.89	151.87	252.46	330.28
Campus	1656.00	420.00	123.00	123.00	25.00	14.00	135.00
Depots	43174.71	3338.00	440.29	440.29	410.00	732.29	1315.14
Driverlog	24999.57	3747.86	813.14	813.14	448.00	329.29	1344.29
DWR	37562.14	3679.14	489.29	489.29	481.43	750.14	1469.00
Easy-IPC-Grid	6595.90	3097.62	1034.82	1034.82	50.23	0.00	977.75
Ferry	15777.29	1190.29	233.86	233.86	162.57	170.14	389.86
Intrusion-Detection	294.33	230.33	102.33	92.33	0.00	34.67	1.00
Kitchen	102.00	71.00	34.00	12.00	0.00	18.00	7.00
Logistics	15595.05	4467.52	991.51	991.51	768.13	10.39	1705.98
Miconic	38783.71	2798.14	928.29	928.29	12.29	6.00	923.29
Rovers	5127.14	1966.43	535.71	517.43	232.14	220.71	460.43
Satellite	10427.29	2116.14	644.43	644.43	67.00	77.86	682.43
Sokoban	16895.14	4600.14	649.43	649.43	698.29	920.29	1682.71
Zeno-Travel	107562.00	6463.00	1104.00	1104.00	1098.00	914.29	2242.71
ALL	20497.93	2619.68	574.28	571.89	305.62	261.16	906.74

BFT, does not visit nodes if their already assigned distance is lower than the current distance. Moreover, an `UNORDERED-AND` node's children are not associated with the prior `ORDERED-AND` node, and thus are visited fewer times than if no `UNORDERED-AND` node is traversed. As a result, for example, although the Action Graph of the Zeno-Travel domain contains more nodes than Driverlog's, its run-time is shorter.

We have identified two ways in which the total processing time of our approach could be reduced. Rather than calculating all the nodes' distances from the goals upfront, this process could be performed for just the observed actions; however, observations would be processed at a reduced rate. Second, the nodes' distances for each goal could be computed in parallel; as we envision this process being performed offline (thus the computational time of this is of lesser importance) and the performance gain would be hardware dependent, this was not implemented.

7.3.3 Results after processing the first N % of observations

Our Action Graph approach outperformed h_{gc} when 10 %, 30 % and 50 % of observations had been received, at 70 % and 100 % h_{gc} slightly outperforms our approach. As described in [40], the lower the number of observations the less likely it is that a landmark is observed; therefore, h_{gc} cannot disambiguate the goals. Fig. 9 displays the average F1-Score, produced by AG3 and h_{gc}, at each percent of observations; Table 4 shows the results per domain for AG1, AG2, AG3 and h_{gc}. The ALL result is the average overall domains rather than problems, so that the result is not weighted towards the domains with the most problems.

Action Graphs have a low precision and recall for the Sokoban domain. GR problems for the Sokoban domain contain observations to navigate to and push two boxes to different locations. The actions for collecting and pushing one box were observed, before the second box was acted on. Whilst observing the actions to push the first box to its goal location, our Action Graph approach increased

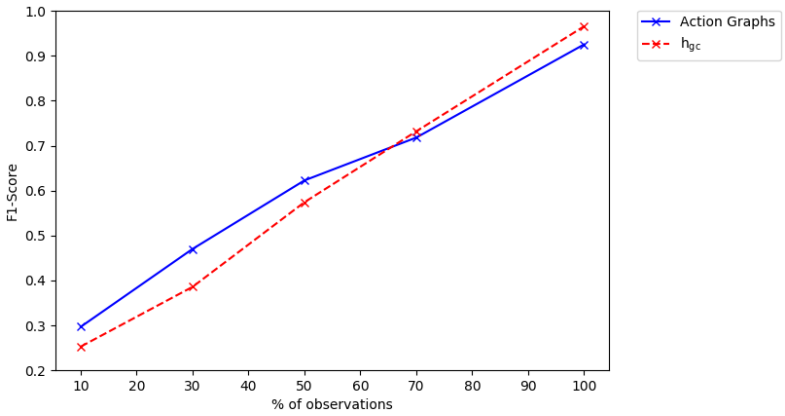


Fig. 9: Average F1-Score, after the first 10, 30, 50, 70 and 100% of observations have been processed, for our Action Graph approach and h_{gc} by Pereira et al. [39].

the probability of the appropriate goals (i.e., the goals the box was becoming closer to). When the observed agent started to navigate to the second box, the aforementioned goals' probability was decreased. Therefore, when the second box is pushed, any of the goals containing a location it is being pushed towards could appear in the set of candidate goals. In other words, the goal probabilities lose information about the first goal atom to be achieved. We considered increasing the probability of the goals with fully observed atoms; however, in problems from other domains some goals' atoms are sub-goals of another goal.

For the Kitchen domain, our Action Graph approach reduced the number of candidate goals significantly more than h_{gc} as few landmarks were observed. On the other hand, due to the structure of the produced graph, Action Graphs produced a low R and M for the Depots and Blocks-World domains. The plans for these domains are highly state dependent, which is not captured by the Action Graph structure. For instance, in a Blocks-World problem, picking up `blockA` requires the gripper to be empty by putting down all blocks (including `blockA`). The graph structure captures the dependencies of actions; however, does not account for the prior state(s) of the environment, e.g., the gripper could already be empty.

AG2 only outperformed AG1 on the Easy-IPC-Grid domain. In this domain there are strong constraints on the order in which actions are performed and a false goal could be traversed on-route to the real goal. Therefore, for this domain, update rule 2 prevented the shortest plan being favoured and successfully increased the probabilities of the goals the observed agent was navigating towards. Nevertheless, this update rule could not determine the real goal for the majority of domains. This is because all goals, whose plans contain the observed action, were multiplied (increased) by the same amount when the current and previous observations were not connected via a `DEP` (or `ORDERED-AND`) node. All suboptimal plans are encoded in an Action Graph's structure; therefore, for many domains, all actions are included within a plan to reach any of the goals. Combining AG2 with AG1 increased the results of the Easy-IPC-Grid domain without greatly affecting the results produced for the other domains. The subsequent sections just show the results of AG3.

Table 4: Accuracy results for the dataset containing the first 10 %, 30 %, 50 %, 70 % and 100 % of observations. AG1 is the first update rule, AG2 the second update rule and AG3 is the combination of the two rules (see Section 6).

			AG1				AG2				AG3				h_{gc}			
Domain	$\|\mathcal{G}\|$	O%	$\|\mathcal{C}\|$	Q	R	M	$\|\mathcal{C}\|$	Q	R	M	$\|\mathcal{C}\|$	Q	R	M	$\|\mathcal{C}\|$	Q	R	M
Blocks-World	20.28	10	9.65	0.52	0.52	0.05	20.01	0.05	1.00	0.05	9.65	0.52	0.52	0.05	1.80	0.87	0.10	0.05
		30	4.08	0.79	0.40	0.11	20.01	0.05	1.00	0.05	4.08	0.79	0.40	0.11	1.22	0.91	0.22	0.18
		50	1.99	0.89	0.36	0.20	20.01	0.05	1.00	0.05	1.99	0.89	0.36	0.20	1.34	0.91	0.32	0.27
		70	1.35	0.94	0.53	0.45	19.99	0.05	1.00	0.05	1.35	0.94	0.53	0.45	1.27	0.94	0.58	0.49
		100	1.10	0.99	0.90	0.87	19.91	0.05	0.99	0.05	1.10	0.99	0.90	0.87	1.36	0.98	1.00	0.86
Campus	2.00	10	1.27	0.87	1.00	0.87	2.00	0.50	1.00	0.50	1.27	0.87	1.00	0.87	1.27	0.60	0.73	0.60
		30	1.00	1.00	1.00	1.00	1.80	0.40	0.80	0.40	1.00	1.00	1.00	1.00	1.07	0.97	1.00	0.97
		50	1.00	0.93	0.93	0.93	1.13	0.07	0.13	0.07	1.00	0.93	0.93	0.93	1.00	1.00	1.00	1.00
		70	1.00	0.80	0.80	0.80	1.13	0.07	0.13	0.07	1.00	0.80	0.80	0.80	1.00	1.00	1.00	1.00
		100	1.00	0.87	0.87	0.87	1.13	0.13	0.20	0.13	1.00	0.80	0.80	0.80	1.00	1.00	1.00	1.00
Depots	8.86	10	5.71	0.40	0.68	0.14	8.86	0.11	1.00	0.11	5.71	0.40	0.68	0.14	1.61	0.75	0.21	0.13
		30	3.32	0.61	0.39	0.22	8.86	0.11	1.00	0.11	3.32	0.61	0.39	0.22	1.71	0.76	0.29	0.21
		50	1.96	0.79	0.54	0.41	8.86	0.11	1.00	0.11	1.96	0.79	0.54	0.41	1.36	0.85	0.50	0.42
		70	1.29	0.89	0.68	0.59	8.86	0.11	1.00	0.11	1.29	0.89	0.68	0.59	1.25	0.93	0.82	0.72
		100	1.14	0.98	1.00	0.96	8.86	0.11	1.00	0.11	1.14	0.98	1.00	0.96	1.04	1.00	1.00	0.98
Driver-log	7.14	10	3.93	0.49	0.71	0.20	7.04	0.16	1.00	0.15	3.93	0.49	0.71	0.20	1.64	0.71	0.29	0.22
		30	2.11	0.73	0.57	0.38	6.61	0.23	1.00	0.18	2.11	0.73	0.57	0.38	1.21	0.80	0.43	0.37
		50	1.82	0.77	0.61	0.51	6.61	0.23	1.00	0.18	1.79	0.79	0.64	0.53	1.21	0.87	0.64	0.55
		70	1.50	0.86	0.75	0.69	6.61	0.23	1.00	0.18	1.50	0.86	0.75	0.69	1.18	0.91	0.75	0.71
		100	1.11	0.96	0.93	0.89	5.54	0.34	1.00	0.24	1.07	0.97	0.93	0.90	1.21	0.97	1.00	0.90
DWR	7.29	10	3.00	0.61	0.57	0.21	7.29	0.14	1.00	0.14	3.00	0.61	0.57	0.21	1.21	0.82	0.43	0.38
		30	1.71	0.78	0.54	0.36	7.29	0.14	1.00	0.14	1.71	0.78	0.54	0.36	1.14	0.87	0.57	0.54
		50	1.25	0.84	0.54	0.43	7.29	0.14	1.00	0.14	1.25	0.84	0.54	0.43	1.11	0.86	0.54	0.50
		70	1.14	0.87	0.57	0.50	7.29	0.14	1.00	0.14	1.14	0.87	0.57	0.50	1.11	0.89	0.64	0.59
		100	1.00	0.99	0.96	0.96	7.29	0.14	1.00	0.14	1.00	0.99	0.96	0.96	1.00	0.98	0.93	0.93
Easy-IPC-Grid	8.36	10	1.64	0.75	0.30	0.21	5.80	0.34	0.74	0.12	1.51	0.76	0.26	0.20	3.62	0.58	0.51	0.20
		30	1.43	0.76	0.25	0.19	3.16	0.61	0.52	0.22	1.52	0.77	0.33	0.27	2.85	0.66	0.44	0.21
		50	1.59	0.76	0.33	0.28	1.85	0.79	0.61	0.46	1.34	0.87	0.61	0.56	2.70	0.69	0.51	0.32
		70	1.20	0.85	0.44	0.44	1.85	0.83	0.72	0.58	1.10	0.89	0.57	0.57	2.57	0.74	0.64	0.45
		100	1.07	0.97	0.87	0.87	1.00	1.00	0.98	0.98	1.00	0.98	0.90	0.90	1.00	1.00	1.00	1.00
Ferry	7.57	10	6.54	0.26	1.00	0.19	7.57	0.13	1.00	0.13	6.54	0.26	1.00	0.19	1.79	0.76	0.54	0.34
		30	1.93	0.87	1.00	0.70	7.57	0.13	1.00	0.13	1.93	0.87	1.00	0.70	1.29	0.89	0.75	0.67
		50	1.25	0.97	1.00	0.88	7.57	0.13	1.00	0.13	1.25	0.97	1.00	0.88	1.04	0.97	0.89	0.88
		70	1.11	0.98	1.00	0.95	7.57	0.13	1.00	0.13	1.11	0.98	1.00	0.95	1.00	0.99	0.96	0.96
		100	1.07	0.99	1.00	0.96	7.57	0.13	1.00	0.13	1.07	0.99	1.00	0.96	1.00	0.99	0.96	0.96
Intrusion-Detection	16.67	10	1.00	0.88	0.07	0.07	6.27	0.68	1.00	0.18	1.00	0.88	0.07	0.07	1.91	0.86	0.31	0.16
		30	2.89	0.88	1.00	0.41	2.89	0.88	1.00	0.41	2.89	0.88	1.00	0.41	1.62	0.87	0.31	0.20
		50	1.51	0.97	1.00	0.78	1.51	0.97	1.00	0.78	1.51	0.97	1.00	0.78	1.40	0.96	0.89	0.70
		70	1.04	1.00	1.00	0.98	1.09	0.99	1.00	0.96	1.04	1.00	1.00	0.98	1.04	1.00	1.00	0.98
		100	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Kitchen	3.00	10	2.00	0.67	1.00	0.63	2.00	0.67	1.00	0.63	2.00	0.67	1.00	0.63	2.87	0.38	1.00	0.38
		30	1.40	0.87	1.00	0.80	1.40	0.87	1.00	0.80	1.40	0.87	1.00	0.80	2.87	0.38	1.00	0.38
		50	1.33	0.89	1.00	0.83	1.33	0.89	1.00	0.83	1.33	0.89	1.00	0.83	2.87	0.38	1.00	0.38
		70	1.33	0.89	1.00	0.83	1.33	0.89	1.00	0.83	1.33	0.89	1.00	0.83	2.33	0.56	1.00	0.56
		100	1.13	0.96	1.00	0.93	1.13	0.96	1.00	0.93	1.13	0.96	1.00	0.93	1.93	0.69	1.00	0.69
Logis-tics	10.40	10	4.41	0.68	1.00	0.31	4.41	0.68	1.00	0.31	4.41	0.68	1.00	0.31	2.80	0.69	0.33	0.15
		30	2.69	0.83	1.00	0.51	2.72	0.83	1.00	0.50	2.69	0.83	1.00	0.51	1.89	0.78	0.33	0.22
		50	1.80	0.92	1.00	0.66	1.90	0.91	1.00	0.64	1.80	0.92	1.00	0.66	1.72	0.86	0.66	0.44
		70	1.67	0.93	1.00	0.71	1.74	0.93	1.00	0.68	1.67	0.93	1.00	0.71	1.51	0.91	0.80	0.58
		100	1.00	1.00	1.00	1.00	1.64	0.94	1.00	0.71	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Miconic	6.00	10	2.96	0.61	0.82	0.31	3.29	0.56	0.82	0.26	3.00	0.61	0.82	0.31	2.00	0.67	0.50	0.27
		30	1.61	0.88	0.93	0.69	1.75	0.82	0.82	0.53	1.61	0.85	0.86	0.62	1.32	0.90	0.86	0.73
		50	1.18	0.97	1.00	0.92	1.21	0.94	0.93	0.83	1.18	0.97	1.00	0.92	1.07	0.94	0.86	0.82
		70	1.07	0.99	1.00	0.96	1.18	0.96	0.96	0.89	1.07	0.99	1.00	0.96	1.04	0.98	0.96	0.95
		100	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.0

Continued on next page...

... Table 4 continued.

			AG1				AG2				AG3				h_{gc}			
Domain	$\|\mathcal{G}\|$	O%	$\|\mathcal{C}\|$	Q	R	M	$\|\mathcal{C}\|$	Q	R	M	$\|\mathcal{C}\|$	Q	R	M	$\|\mathcal{C}\|$	Q	R	M
Rovers	6.00	10	3.64	0.54	0.93	0.32	4.61	0.40	1.00	0.30	3.64	0.54	0.93	0.32	2.00	0.65	0.46	0.24
		30	1.61	0.88	0.93	0.70	2.25	0.77	0.93	0.62	1.61	0.88	0.93	0.70	1.14	0.88	0.71	0.66
		50	1.11	0.96	0.93	0.88	1.64	0.89	1.00	0.73	1.11	0.96	0.93	0.88	1.14	0.94	0.89	0.83
		70	1.07	0.99	1.00	0.96	1.18	0.97	1.00	0.91	1.07	0.99	1.00	0.96	1.11	0.97	0.96	0.93
		100	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Satellite	6.43	10	4.00	0.51	1.00	0.39	6.25	0.19	1.00	0.16	4.00	0.51	1.00	0.39	2.11	0.71	0.68	0.42
		30	3.43	0.60	1.00	0.49	5.36	0.33	1.00	0.25	3.43	0.60	1.00	0.49	1.96	0.76	0.75	0.51
		50	1.43	0.90	0.89	0.71	2.21	0.81	1.00	0.64	1.43	0.90	0.89	0.71	1.21	0.92	0.86	0.77
		70	1.11	0.93	0.86	0.80	1.46	0.92	1.00	0.85	1.11	0.93	0.86	0.80	1.04	0.96	0.89	0.88
		100	1.04	0.99	1.00	0.98	1.11	0.98	1.00	0.95	1.04	0.99	1.00	0.98	1.07	0.99	1.00	0.96
Sokoban	7.14	10	1.57	0.68	0.18	0.12	7.00	0.15	1.00	0.15	1.57	0.68	0.18	0.12	2.36	0.63	0.43	0.17
		30	1.14	0.77	0.32	0.30	6.82	0.16	0.96	0.14	1.14	0.77	0.32	0.30	1.89	0.66	0.32	0.20
		50	1.11	0.81	0.39	0.38	6.82	0.16	0.96	0.14	1.11	0.81	0.39	0.38	1.21	0.87	0.64	0.57
		70	1.04	0.81	0.39	0.39	6.82	0.16	0.96	0.14	1.04	0.81	0.39	0.39	1.14	0.91	0.75	0.70
		100	1.04	0.84	0.50	0.50	6.64	0.18	0.93	0.13	1.04	0.84	0.50	0.50	1.00	1.00	1.00	1.00
Zeno-Travel	6.86	10	4.14	0.49	0.79	0.32	6.86	0.15	1.00	0.15	4.14	0.49	0.79	0.32	1.50	0.73	0.32	0.24
		30	2.96	0.66	0.71	0.43	6.86	0.15	1.00	0.15	2.96	0.66	0.71	0.43	1.61	0.77	0.54	0.38
		50	1.82	0.84	0.82	0.61	6.79	0.16	1.00	0.15	1.82	0.84	0.82	0.61	1.36	0.89	0.79	0.70
		70	1.14	0.96	0.93	0.86	6.79	0.16	1.00	0.15	1.14	0.96	0.93	0.86	1.11	0.97	0.96	0.95
		100	1.00	1.00	1.00	1.00	6.57	0.20	1.00	0.16	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
ALL	8.26	10	3.70	0.60	0.70	0.29	6.62	0.33	0.97	0.22	3.69	0.60	0.70	0.29	2.03	0.69	0.46	0.26
		30	2.22	0.79	0.74	0.49	5.69	0.43	0.94	0.31	2.23	0.79	0.74	0.49	1.65	0.79	0.57	0.43
		50	1.48	0.88	0.76	0.63	5.12	0.48	0.91	0.39	1.46	0.89	0.78	0.65	1.45	0.86	0.73	0.61
		70	1.20	0.91	0.80	0.73	4.99	0.50	0.92	0.44	1.20	0.92	0.81	0.74	1.31	0.91	0.85	0.76
		100	1.05	0.97	0.94	0.92	4.76	0.54	0.94	0.51	1.04	0.97	0.93	0.92	1.11	0.97	0.99	0.95

7.3.4 Missing observations results discussion

Our Action Graph approach and h_{gc} were also ran on the 6313 GR problems[4] produced by Pereira et al. [39] and Ramírez and Geffner [44], that contain missing observations. These problems contain a random 10, 30, 50, 70 and 100 % of observations. The F1-Scores are depicted in Fig. 10 and a table containing the results per domain can be found at `https://doi.org/10.5281/zenodo.3621275`.

These results show a similar trend to the previous experiment, i.e., our approach produced a higher F1-Score than h_{gc} at 10 %, 30 % and 50 % of observations (and vice versa after 70 % and 100 % of observations). Both approaches perform better on the dataset containing missing observations, than they did in the previous experiment, as each GR problem could contain observations that are close to the goal (due to random actions in a plan having been selected).

8 Related work

Methods for intention recognition can be broadly categorised as data-driven and knowledge-driven (i.e., symbolic) methods [42, 59]. Data-driven approaches train a recognition model from a large dataset [1, 3, 50, 59]. The main disadvantages of this method are that often a large amount of labelled training data is required and the produced models often only work on data similar to the training set [46, 58]. Since our work belongs to the category of knowledge-driven methods, data-driven methods are not further discussed.

Knowledge-driven approaches rely on a logical description of the actions agents can perform. They can be further divided into approaches that parse a library of

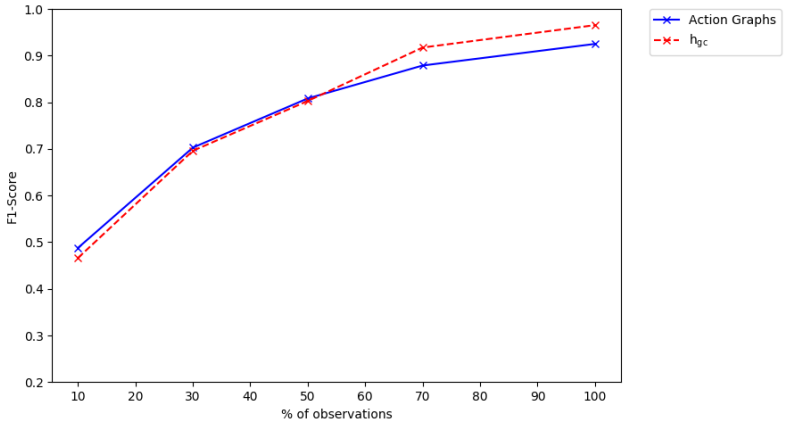


Fig. 10: Average F1-Score produced by our Action Graph approach and h_{gc} by Pereira et al. [39] on the dataset containing missing observations (in other words when 10, 30, 50, 70 and 100 % of actions had been observed).

plans (also known as "recognition as parsing"), and approaches that solve recognition problems, defined in languages usually associated with planning, i.e., "recognition as planning" [29]. Our GR approach derives a graph structure, similar to those used by some recognition as parsing methods, from a PDDL defined (planning-based) GR problem. Recognition as planning is often viewed as more flexible and general because a library of plans is not required and cyclic plans are difficult to compile into a library [44]. We chose to transform a PDDL planning problem into an Action Graph to enable the goal probabilities to be updated quickly, all plans (including suboptimal plans) to be represented, cyclic plans to be expressed and inaccurate initial states to be handle. Our approach takes advantage of the fact that a perfect/complete representation of plans is not required to perform GR. In this section, recognition as parsing and recognition as planning approaches are discussed in turn.

8.1 Recognition as parsing

In recognition as parsing, hierarchical structures are usually developed which include abstract actions along with how they are decomposed to concrete (observable) actions [28]. Several prior approaches have represented these hierarchical structures as AND/OR trees [14, 21]. As previously mentioned, our graph structure was inspired by these works. The recognition as parsing approaches, mentioned in this section, enable both the goal and plan of the observed agent to be recognised but do not mention handling invalid initial states or suboptimal plans.

Kautz et al. [27, 28] introduce a language to describe a hierarchy of actions. Based on which low level actions are observed, the higher level task(s) an agent is attempting to achieve is inferred. Their paper presents one of the earliest plan/goal recognition formal theories that aimed to handle simultaneous action execution, multi-plan recognition and missing observations.

A set of action sequence graphs is derived from a library of plans in [54]. This set is compared to an action sequence graph, created from a sequence of observations, to find the plan most similar to the observation sequence. Their approach was shown to perform well on misclassified (incorrect) sensor observations and missing actions; but to generate the library of plans a planner is called, and thus a known initial state is required.

8.2 Recognition as planning

Recognition as planning is a more recently proposed approach, in which languages normally associated with task planning, such as STRIPS [9] and PDDL [35], define the actions agents can perform (along with their preconditions and effects) and world states. In recognition as parsing there are usually only action definitions, whereas planning-based approaches allow for the inclusion of state knowledge, such as what objects are found within the environment and their locations.

In [43, 44] it was proposed to view goal recognition as the inverse of planning. To find the difference in the cost of the plan to reach the goal with and without taking the observations into consideration, a planner is called twice for every possible goal. Therefore, the performance would greatly deteriorate when exposed to inaccurate initial states. In [4] the work from [44] was extended, to find the joint probability of pairs of goals rather than a single goal. Their work aimed to handle multiple interleaving goals. Although initial approaches were computationally expensive as they required a task planner to be called multiple times [4, 43, 44], the latest advances in recognition as planning algorithms have greatly improved this [7, 39].

Plan graphs were proposed in [7]. A plan graph, which contains actions and propositions labelled as either true, false or unknown, is built from a planning problem and updated based on the observations. Rather than calling a planner, the graph is used to calculate the cost of reaching the goals. Our Action Graph structure differs greatly from a plan graph, as Action Graphs only contain actions and the constraints between those actions.

More recently Pereira et al. [39, 40] significantly reduced the recognition time by finding landmarks. Our experiments show a comparison to this approach. This work has been expanded to handle incomplete domain models [41], i.e., GR problems with incomplete preconditions and effects. In future work, we will explore applying our work to incomplete domain models.

9 Conclusion

Our novel approach to goal recognition aims to handle problems in which the defined initial state is inaccurate. An inaccurate initial state contains fluents whose value is unknown and/or incorrect. For instance, if an item or agent (e.g., cup or human) is occluded its location is indeterminable, and thus possibly defined incorrectly. Our approach transforms a PDDL defined GR problem into an Action Graph, which models the order constraints between actions. Each node is labelled with the minimum number of `DEP` and `ORDERED-AND` nodes, traversed to reach it from each goal. When an action is observed, the goals' probability is updated

based on either the distance the action's associated node is from the goals or, if the current and prior observation are connected via a `DEP` or `ORDERED-AND` node, the change in distance. Experiments proved that when the fluents have incorrect values in the initial state, e.g., because they are unknown or sensed/determined incorrectly, the performance of our approach is unaffected.

In future work, we intend to apply our Action Graph method to further challenges associated with symbolic GR. As well as the defined initial state being inaccurate, the domain model (i.e., action definitions) could be incorrect [41]. Therefore, we will experiment with adapting the Action Graph structure based on the order observations are received. To create a more compact structure, and thus reduce the computational time of this, we will investigate grouping related actions into a single node. For instance, in the Sokoban domain, the same actions can be performed on both `box1` and `box2`; therefore, actions, such as `push(box1 loc1 loc2)` and `push(box2 loc1 loc2)`, can be grouped into a single node. Moreover, we intend to apply our GR approach to problems in which either the observed agent has multiple goals, or multiple agents have individual or joint goals [25].

Another direction for future research is to modify our approach so that the Action Graph structure expands over time. As more observation are made, new actions could be inserted and the links between actions could be adjusted. This could enable actions and sequences to be learnt. The performance of our current method could be compared to this and to recurrent neural networks (RNN) based approaches [3] using real world data.

As developing the PDDL can be time consuming and challenging, researches have attempted to replace this manual process, with deep learning methods [1, 2]. We will explore the potential of learning the Action Graph structure from pairs of images, and then converting the Action Graph into a PDDL defined domain. Which, subsequently, could be provided as input to task planners as well as goal recognisers.

Acknowledgements The research leading to this article was funded through an SB Fellowship of the FWO-Vlaanderen (project no. 1S40217N). Helen Harman is currently affiliated with the University of Lincoln.

Conflict of interest

The authors declare that they have no conflict of interest.

Appendix

A Action Graph creation algorithm

The pseudo-code for our Action Graph creation method, described in Section 4, is provided in Algorithm 2.

B Algorithm to the nodes' distance from each goal

The pseudo-code for the BFT that labels the nodes with their distance from each goal, described in Section 5, is provided in Algorithm 3.

Algorithm 2: Action Graph creation

Data: $\mathcal{A}$ set of all actions, $\mathcal{G}$ set of hypothesis goals
Result: Action Graph

$root_n \leftarrow ORnode(\emptyset)$ ▷ Create OR node with no children
foreach $a \in \mathcal{A}$ **do**
  $\mathcal{D}_a \leftarrow [\emptyset]$ ▷ 2D array - array per precondition
  foreach $p \in a_{pre}$ **do** $\mathcal{D}_a$ += $\{a' \mid p \in a'_{eff}\}$ ▷ Get dependencies
  if $\mathcal{D}_a = [\emptyset]$ **then** ▷ e.g. because a has 0 preconditions
    $root_n.children$ += $node(a)$ ▷ append a's Action Node to root
    continue
  end
  $dep_n \leftarrow DEPnode(\emptyset)$ ▷ Create DEP node
  $root_n.children$ += dep_n
  if $|\mathcal{D}_a| = 1$ **then** ▷ e.g. because a has 1 precondition
    $dep_n.children = [ORnode(\mathcal{D}_a[0]), node(a)]$
    continue
  end
  $u\text{-}and_n \leftarrow UANDnode(\emptyset)$ ▷ Create UNORDERED-AND node
  $dep_n.children = (u\text{-}and_n, node(a))$
  foreach $D_p \in \mathcal{D}_a$ **do**
    $or_n = ORnode(\mathcal{D}_p)$ ▷ Actions that set the same precondition are always the children of an (the same) OR node
    if $\{a' \mid a' \in D_p, a'$ *is affected by any* $\{a'' \mid a'' \in D'_p, D'_p \in \mathcal{D}_a\}\ \} \neq \emptyset$ ***and*** $\mathcal{D}_a$ *does not have cyclic order constraints* **then**
      $o\text{-}and_n \leftarrow OANDnode(or_n,$ affecting actions$)$ ▷ Add O-AND node, and U-AND node when >1 affecting actions
      $u\text{-}and_n.children$ += $o\text{-}and_n$
    else if D_p *not already decedents of* $u\text{-}and_n$ **then**
      $u\text{-}and_n.children$ += or_n
    end
  end
end
$createGoalActionsAndSetRootsChildren(G \in \mathcal{G}, root.children)$

C Domains

The list below describes each domain in turn, and describes the fluents whose initial value was modified to generate the dataset for the experiments described in Section 7.2. Further details on the modified fluents are provided at `https://doi.org/10.5281/zenodo.3621275`. These GR domains were produced by Ramírez and Geffner [44], Pereira et al. [39][4], based on the work of Wu et al. [57] and the IPC domains[1].

- `Blocks-World`: A hand stacks blocks on top of one another to create a tower.
  - A block can be placed on the table or another block, a block can become unclear, and the hand can be empty or holding a block.
- `Campus`: In the campus domain a university student navigates to different locations (e.g., the library, a cafe, ect.) to perform different activities.
  - The student's location can change and if an activity has been performed (or not) can be modified.
- `Depots`: In this domain, crates are relocated by trucks and hoists.
  - Crates can change location, a crate could be on another create or in a truck, and a hoist could be lifting a crate or available.
- `Driverlog`: Trucks are driven by different drives, so that objects can be relocated.
  - Trucks, drivers and objects can change location, an object could be in a truck, and a driver can be driving a truck.
- `DWR`: Robots and cranes relocate containers, which are piled on top of each other.
  - The location of a robot, which pile a container is in (and/or on top of), if a container has been loaded onto a robot and if a crane is holding a container can be changed.

Algorithm 3: Nodes' distance from each goal initialisation

```
   Data: A_g set of goal action nodes, Action Graph
   Result: Action Graph with nodes' distance from each goal
 1 foreach a_g ∈ A_g do setNodeValueBFT(a_g.parent, 0, a_g.goal)
 2 Function setNodeValueBFT(node, startCount, G)
 3     queue = ∅
 4     queue.push(BftNode(node, startCount, null))
 5     while queue ≠ ∅ do
 6         currentNode, count, o-and = queue.pop()
 7         if currentNode.getDisFromGoal(G, o-and) ≤ count then
 8             continue
 9         currentNode.setDisFromGoal(count, G, o-and)
10         if currentNode.type is DEPnode() or OANDnode() then
11             count = count + 1
12         foreach child ∈ currentNode.children do
13             if child.getDisFromGoal(G, o-and) ≤ count then continue
14             if child.type is actionNode then
15                 child.setDisFromGoal(count, G, o-and)
16             else
17                 if child.type is UANDnode() then
18                     queue.push(BftNode(child, count, null))
19                 else if currentNode.type is OANDnode() and
                    currentNode.children[1] = child then
20                     queue.push(BftNode(child, count, currentNode))
21                 else
22                     queue.push(BftNode(child, count, o-and))
23                 end
24             end
25         end
26     end
27 end
```

- `Easy-IPC-Grid`: A robot navigates a grid to reach a goal location, and along the way must collect keys to unlock locations.
  - A location can be unlocked, and the location of the robot and if a key is being carried can be modified.
- `Ferry`: A ferry transports cars to different locations.
  - The ferry's location and if a car is at a location or on the ferry can be changed.
- `Intrusion-Detection`: An intruder accesses, modifies and downloads information from a computer system.
  - What information has been accessed, modified or downloaded can be changed.
- `Kitchen`: The observed agent takes different items and performs kitchen-based activities (e.g., makes toast). Note that: "activities" are not included in the list of observations.
  - Which items have been taken, what equipment has been used and what activities have been performed can be changed.
- `Logistics`: Packages are transported to different locations and airports by air planes and trucks.
  - Which airport, location, truck or airplane a package is at/in; a truck's airport/location, and an airplane's airport can be changed.
- `Miconic`: A lift takes different passages to there desired floor.
  - The floor the lift starts on, which passages are in the lift and which passages have been served can be changed.
- `Rovers`: A rover navigates a planet collecting rock samples, soil samples and images.
  - The location of the rover, if a camera has been calibrated, if the robots store is empty/full, what data has been collected, which data has been communicated and if a channel is free can be modified.
- `Satellite`: In the satellite domain, satellites take images in different modes and directions.

  - Which direction a satellite is pointing in, if the power is being availed, if the power is on, if the an instrument is calibrated and if an image has been taken can be modified.
- `Sokoban`: A robot must push two boxes to different locations.
  - The location of the robot and the boxes can be modified.
- `Zeno-travel`: People travel on aircraft, to reach different cities.
  - Which city/aircraft a person is in/at, which city an aircraft is in and an aircraft's fuel level can be changed.

D Inaccurate initial state detailed experimental results

As described in Section 7.2, experiments where performed in which varying amounts of fluents were set to incorrect values. A breakdown of the results, per domain, for these experiments are provided in Tables 5 and 6.

Table 5: Effect increasing the amount of incorrect fluents in the initial state had on the accuracy of our Action Graph approach and on h_{gc} [39] per domain.

			ANY %				20 %				40 %			
			Action Graphs				h_{gc}				h_{gc}			
Domain	$\|\mathcal{G}\|$	O%	$\|\mathcal{C}\|$	Q	R	M	$\|\mathcal{C}\|$	Q	R	M	$\|\mathcal{C}\|$	Q	R	M
Blocks-World	20.28	10	9.65	0.52	0.52	0.05	3.24	0.81	0.20	0.09	4.73	0.74	0.30	0.09
		30	4.08	0.79	0.40	0.11	3.38	0.81	0.26	0.10	4.95	0.74	0.37	0.11
		50	1.99	0.89	0.36	0.20	3.35	0.82	0.33	0.13	5.95	0.70	0.43	0.11
		70	1.35	0.94	0.53	0.45	3.42	0.82	0.46	0.23	5.20	0.74	0.52	0.16
		100	1.10	0.99	0.90	0.87	3.86	0.85	0.97	0.52	6.26	0.71	0.77	0.32
Campus	2.00	10	1.27	0.87	1.00	0.87	1.07	0.57	0.60	0.57	1.00	0.47	0.47	0.47
		30	1.00	1.00	1.00	1.00	1.00	0.73	0.73	0.73	1.00	0.53	0.53	0.53
		50	1.00	0.93	0.93	0.93	1.07	0.77	0.80	0.77	1.07	0.83	0.87	0.83
		70	1.00	0.80	0.80	0.80	1.07	0.90	0.93	0.90	1.13	0.47	0.53	0.47
		100	1.00	0.80	0.80	0.80	1.07	0.90	0.93	0.90	1.07	0.97	1.00	0.97
Depots	8.86	10	5.71	0.40	0.68	0.14	2.18	0.70	0.25	0.12	2.71	0.66	0.32	0.10
		30	3.32	0.61	0.39	0.22	2.07	0.71	0.25	0.09	2.25	0.71	0.32	0.20
		50	1.96	0.79	0.54	0.41	2.18	0.75	0.46	0.24	3.14	0.61	0.29	0.06
		70	1.29	0.89	0.68	0.59	1.79	0.79	0.46	0.33	2.71	0.69	0.46	0.25
		100	1.14	0.98	1.00	0.96	1.89	0.77	0.39	0.33	3.21	0.63	0.43	0.15
Driverlog	7.14	10	3.93	0.49	0.71	0.20	1.14	0.76	0.21	0.20	1.32	0.71	0.18	0.15
		30	2.11	0.73	0.57	0.38	1.29	0.78	0.36	0.30	1.21	0.82	0.50	0.44
		50	1.79	0.79	0.64	0.53	1.36	0.76	0.39	0.31	1.36	0.79	0.46	0.33
		70	1.50	0.86	0.75	0.69	1.04	0.89	0.64	0.64	1.50	0.79	0.50	0.45
		100	1.07	0.97	0.93	0.90	1.11	0.92	0.79	0.76	1.25	0.90	0.75	0.71
DWR	7.29	10	3.00	0.61	0.57	0.21	1.36	0.77	0.32	0.29	1.29	0.77	0.32	0.26
		30	1.71	0.78	0.54	0.36	1.11	0.82	0.39	0.38	1.32	0.72	0.14	0.13
		50	1.25	0.84	0.54	0.43	1.14	0.78	0.29	0.24	1.29	0.76	0.25	0.20
		70	1.14	0.87	0.57	0.50	1.14	0.82	0.43	0.43	1.25	0.81	0.43	0.35
		100	1.00	0.99	0.96	0.96	1.07	0.90	0.68	0.66	1.07	0.84	0.46	0.45
Easy-IPC-Grid	8.36	10	1.51	0.76	0.26	0.20	3.69	0.55	0.48	0.16	3.59	0.56	0.46	0.15
		30	1.52	0.77	0.33	0.27	2.98	0.65	0.46	0.21	2.93	0.61	0.33	0.10
		50	1.34	0.87	0.61	0.56	2.57	0.67	0.41	0.23	2.75	0.63	0.33	0.16
		70	1.10	0.89	0.57	0.57	2.61	0.71	0.52	0.33	2.67	0.69	0.49	0.29
		100	1.00	0.98	0.90	0.90	1.02	0.98	0.95	0.94	1.00	1.00	1.00	1.00

Continued on next page...

... Table 5 continued.

			ANY %				20 %				40 %			
			Action Graphs				h_{gc}				h_{gc}			
Domain	$\|\mathcal{G}\|$	O%	$\|\mathcal{C}\|$	Q	R	M	$\|\mathcal{C}\|$	Q	R	M	$\|\mathcal{C}\|$	Q	R	M
		10	6.54	0.26	1.00	0.19	1.57	0.73	0.29	0.16	1.29	0.76	0.25	0.21
		30	1.93	0.87	1.00	0.70	1.39	0.87	0.75	0.65	1.14	0.83	0.43	0.40
Ferry	7.57	50	1.25	0.97	1.00	0.88	1.04	0.97	0.93	0.91	1.11	0.92	0.75	0.71
		70	1.11	0.98	1.00	0.95	1.00	0.99	0.96	0.96	1.04	0.97	0.93	0.91
		100	1.07	0.99	1.00	0.96	1.00	0.99	0.96	0.96	1.00	0.99	0.96	0.96
		10	1.00	0.88	0.07	0.07	1.31	0.85	0.02	0.01	1.93	0.83	0.13	0.09
		30	2.89	0.88	1.00	0.41	1.04	0.89	0.18	0.16	1.69	0.84	0.09	0.07
Intrusion-Detection	16.67	50	1.51	0.97	1.00	0.78	1.20	0.90	0.36	0.31	1.44	0.86	0.13	0.09
		70	1.04	1.00	1.00	0.98	1.18	0.92	0.44	0.41	1.80	0.85	0.24	0.11
		100	1.00	1.00	1.00	1.00	1.04	0.97	0.76	0.73	1.42	0.86	0.16	0.14
		10	2.00	0.67	1.00	0.63	2.13	0.49	0.80	0.41	2.00	0.44	0.67	0.30
		30	1.40	0.87	1.00	0.80	1.40	0.64	0.67	0.52	1.40	0.51	0.47	0.34
Kitchen	3.00	50	1.33	0.89	1.00	0.83	1.60	0.58	0.67	0.47	1.53	0.51	0.53	0.33
		70	1.33	0.89	1.00	0.83	1.60	0.49	0.53	0.37	1.73	0.36	0.40	0.17
		100	1.13	0.96	1.00	0.93	1.47	0.80	0.93	0.74	1.73	0.53	0.67	0.38
		10	4.41	0.68	1.00	0.31	1.72	0.79	0.26	0.13	1.74	0.78	0.20	0.08
		30	2.69	0.83	1.00	0.51	1.67	0.80	0.30	0.19	1.36	0.83	0.30	0.20
Logistics	10.40	50	1.80	0.92	1.00	0.66	1.54	0.86	0.52	0.37	1.48	0.82	0.33	0.23
		70	1.67	0.93	1.00	0.71	1.41	0.89	0.66	0.47	1.33	0.88	0.56	0.42
		100	1.00	1.00	1.00	1.00	1.03	0.96	0.79	0.78	1.26	0.93	0.75	0.69
		10	3.00	0.61	0.82	0.31	1.21	0.74	0.32	0.29	1.21	0.68	0.14	0.10
		30	1.61	0.85	0.86	0.62	1.21	0.85	0.64	0.57	1.39	0.74	0.43	0.30
Miconic	6.00	50	1.18	0.97	1.00	0.92	1.04	0.91	0.75	0.75	1.11	0.82	0.50	0.46
		70	1.07	0.99	1.00	0.96	1.11	0.96	0.93	0.88	1.18	0.82	0.54	0.46
		100	1.00	1.00	1.00	1.00	1.11	0.98	1.00	0.95	1.61	0.90	1.00	0.82
		10	3.64	0.54	0.93	0.32	1.46	0.66	0.21	0.14	2.89	0.54	0.57	0.21
		30	1.61	0.88	0.93	0.70	1.75	0.68	0.43	0.31	3.07	0.55	0.68	0.31
Rovers	6.00	50	1.11	0.96	0.93	0.88	1.36	0.85	0.71	0.64	2.89	0.59	0.71	0.37
		70	1.07	0.99	1.00	0.96	2.07	0.74	0.75	0.50	2.82	0.59	0.68	0.36
		100	1.00	1.00	1.00	1.00	1.82	0.73	0.61	0.51	2.96	0.58	0.71	0.38
		10	4.00	0.51	1.00	0.39	1.57	0.69	0.32	0.23	1.25	0.77	0.43	0.39
		30	3.43	0.60	1.00	0.49	1.29	0.74	0.32	0.27	1.36	0.72	0.32	0.27
Satellite	6.43	50	1.43	0.90	0.89	0.71	1.07	0.85	0.57	0.55	1.18	0.83	0.57	0.52
		70	1.11	0.93	0.86	0.80	1.11	0.91	0.79	0.77	1.32	0.89	0.82	0.70
		100	1.04	0.99	1.00	0.98	1.11	0.97	0.96	0.92	1.46	0.91	0.96	0.80
		10	1.57	0.68	0.18	0.12	2.36	0.63	0.43	0.17	2.39	0.60	0.32	0.09
		30	1.14	0.77	0.32	0.30	1.89	0.66	0.32	0.20	1.71	0.67	0.25	0.20
Sokoban	7.14	50	1.11	0.81	0.39	0.38	1.21	0.87	0.64	0.57	1.14	0.81	0.39	0.34
		70	1.04	0.81	0.39	0.39	1.14	0.91	0.75	0.70	1.29	0.87	0.64	0.55
		100	1.04	0.84	0.50	0.50	1.00	1.00	1.00	1.00	1.04	0.96	0.89	0.88
		10	4.14	0.49	0.79	0.32	1.46	0.72	0.32	0.20	1.14	0.73	0.18	0.18
		30	2.96	0.66	0.71	0.43	1.68	0.70	0.32	0.21	1.46	0.77	0.43	0.31
Zeno-Travel	6.86	50	1.82	0.84	0.82	0.61	1.36	0.88	0.75	0.68	1.29	0.88	0.71	0.66
		70	1.14	0.96	0.93	0.86	1.18	0.95	0.93	0.89	1.14	0.91	0.75	0.71
		100	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.04	0.98	0.96	0.95
		10	3.69	0.60	0.70	0.29	1.83	0.70	0.34	0.21	2.03	0.67	0.33	0.19
		30	2.23	0.79	0.74	0.49	1.68	0.76	0.43	0.33	1.88	0.71	0.37	0.26
ALL	8.26	50	1.46	0.89	0.78	0.65	1.54	0.81	0.57	0.48	1.91	0.76	0.48	0.36
		70	1.20	0.92	0.81	0.74	1.52	0.85	0.68	0.59	1.87	0.75	0.57	0.42
		100	1.04	0.97	0.93	0.92	1.37	0.92	0.85	0.78	1.83	0.85	0.77	0.64

Table 6: Effect increasing the amount of incorrect fluents in the initial state had on the accuracy of h_{gc} [39], continued from Table 5.

			60 %				80 %				100 %			
			h_{gc}				h_{gc}				h_{gc}			
Domain	$\|\mathcal{G}\|$	O%	$\|\mathcal{C}\|$	Q	R	M	$\|\mathcal{C}\|$	Q	R	M	$\|\mathcal{C}\|$	Q	R	M
Blocks-World	20.28	10	5.71	0.69	0.29	0.07	5.00	0.73	0.27	0.07	3.27	0.81	0.22	0.08
		30	5.14	0.72	0.29	0.07	4.80	0.74	0.26	0.08	3.85	0.78	0.21	0.07
		50	6.21	0.68	0.45	0.12	4.98	0.74	0.36	0.10	3.76	0.79	0.26	0.10
		70	5.95	0.70	0.43	0.14	5.79	0.69	0.33	0.12	4.03	0.78	0.35	0.14
		100	6.48	0.69	0.67	0.32	5.96	0.71	0.58	0.27	5.63	0.72	0.54	0.26
Campus	2.00	10	1.00	0.53	0.53	0.53	1.00	0.40	0.40	0.40	2.00	0.50	1.00	0.50
		30	1.07	0.70	0.73	0.70	1.00	0.47	0.47	0.47	2.00	0.50	1.00	0.50
		50	1.07	0.50	0.53	0.50	1.00	0.53	0.53	0.53	2.00	0.50	1.00	0.50
		70	1.00	0.53	0.53	0.53	1.00	0.60	0.60	0.60	2.00	0.50	1.00	0.50
		100	1.27	0.80	0.93	0.80	1.07	0.57	0.60	0.57	2.00	0.50	1.00	0.50
Depots	8.86	10	2.79	0.68	0.46	0.13	2.96	0.63	0.32	0.06	3.71	0.55	0.36	0.11
		30	2.68	0.66	0.32	0.11	2.96	0.63	0.29	0.07	2.32	0.68	0.18	0.07
		50	2.39	0.68	0.25	0.11	2.32	0.67	0.18	0.08	3.11	0.61	0.29	0.06
		70	2.00	0.73	0.29	0.11	2.54	0.66	0.21	0.06	2.39	0.67	0.25	0.11
		100	2.36	0.68	0.18	0.12	2.64	0.68	0.36	0.12	2.93	0.63	0.32	0.07
Driverlog	7.14	10	1.54	0.70	0.21	0.15	1.68	0.71	0.32	0.20	1.54	0.68	0.14	0.08
		30	1.36	0.70	0.14	0.07	1.54	0.70	0.25	0.19	1.79	0.68	0.25	0.15
		50	1.50	0.71	0.25	0.22	1.79	0.68	0.25	0.16	1.50	0.71	0.25	0.13
		70	1.46	0.83	0.61	0.51	1.68	0.73	0.39	0.28	1.93	0.62	0.14	0.07
		100	1.43	0.87	0.75	0.64	1.57	0.77	0.46	0.36	1.89	0.74	0.54	0.35
DWR	7.29	10	1.14	0.76	0.21	0.16	1.39	0.73	0.21	0.15	1.11	0.72	0.04	0.04
		30	1.07	0.74	0.11	0.07	1.11	0.76	0.18	0.16	1.21	0.74	0.18	0.18
		50	1.04	0.79	0.29	0.27	1.18	0.76	0.25	0.25	1.32	0.73	0.18	0.12
		70	1.14	0.77	0.25	0.21	1.14	0.72	0.07	0.05	1.07	0.79	0.29	0.27
		100	1.11	0.85	0.50	0.48	1.18	0.82	0.46	0.40	1.29	0.78	0.32	0.27
Easy-IPC-Grid	8.36	10	3.46	0.56	0.44	0.14	3.59	0.57	0.49	0.16	3.34	0.57	0.44	0.14
		30	2.62	0.64	0.33	0.13	2.49	0.68	0.39	0.21	2.46	0.64	0.25	0.11
		50	2.66	0.66	0.39	0.22	2.59	0.64	0.31	0.15	2.41	0.66	0.28	0.13
		70	2.51	0.69	0.44	0.24	2.66	0.67	0.43	0.26	2.56	0.67	0.38	0.22
		100	1.02	1.00	1.00	0.99	1.10	0.98	0.97	0.92	1.10	0.99	1.00	0.95
Ferry	7.57	10	1.29	0.76	0.25	0.20	1.25	0.76	0.25	0.20	1.54	0.74	0.29	0.20
		30	1.00	0.88	0.57	0.57	1.11	0.83	0.46	0.43	1.18	0.82	0.43	0.38
		50	1.11	0.89	0.64	0.61	1.07	0.90	0.64	0.61	1.21	0.85	0.54	0.45
		70	1.11	0.95	0.89	0.86	1.07	0.90	0.68	0.64	1.14	0.86	0.57	0.52
		100	1.00	0.98	0.93	0.93	1.00	0.98	0.93	0.93	1.07	0.92	0.75	0.73
Intrusion-Detection	16.67	10	3.49	0.76	0.29	0.09	8.33	0.51	0.58	0.08	16.67	0.07	1.00	0.07
		30	3.36	0.76	0.27	0.07	8.91	0.48	0.60	0.07	16.67	0.07	1.00	0.07
		50	3.53	0.75	0.16	0.05	8.13	0.52	0.53	0.08	16.67	0.07	1.00	0.07
		70	4.31	0.69	0.20	0.05	8.27	0.49	0.42	0.06	16.67	0.07	1.00	0.07
		100	2.93	0.78	0.18	0.06	7.13	0.54	0.36	0.05	16.67	0.07	1.00	0.07
Kitchen	3.00	10	2.13	0.44	0.73	0.32	2.53	0.36	0.80	0.30	3.00	0.33	1.00	0.33
		30	2.27	0.40	0.73	0.32	1.93	0.51	0.73	0.40	3.00	0.33	1.00	0.33
		50	1.93	0.47	0.67	0.32	2.47	0.42	0.87	0.39	3.00	0.33	1.00	0.33
		70	1.87	0.44	0.60	0.32	2.13	0.36	0.60	0.28	3.00	0.33	1.00	0.33
		100	1.87	0.44	0.60	0.32	2.33	0.33	0.67	0.26	3.00	0.33	1.00	0.33
Logistics	10.40	10	1.75	0.76	0.15	0.06	1.79	0.76	0.18	0.07	2.11	0.75	0.25	0.15
		30	1.75	0.78	0.26	0.12	1.80	0.77	0.21	0.12	1.93	0.75	0.16	0.10
		50	1.75	0.77	0.20	0.13	1.52	0.80	0.25	0.17	1.89	0.77	0.28	0.14
		70	1.41	0.82	0.28	0.22	1.59	0.80	0.28	0.19	1.66	0.78	0.18	0.12
		100	1.11	0.93	0.70	0.69	1.28	0.90	0.59	0.53	1.43	0.89	0.64	0.54
Miconic	6.00	10	1.36	0.70	0.29	0.23	2.00	0.63	0.39	0.20	6.00	0.17	1.00	0.17
		30	1.21	0.81	0.54	0.47	1.68	0.65	0.29	0.20	6.00	0.17	1.00	0.17
		50	1.32	0.77	0.46	0.39	2.00	0.65	0.46	0.24	6.00	0.17	1.00	0.17
		70	1.39	0.79	0.57	0.54	2.64	0.61	0.64	0.29	6.00	0.17	1.00	0.17
		100	1.96	0.84	1.00	0.72	3.07	0.65	1.00	0.51	6.00	0.17	1.00	0.17
Rovers	6.00	10	5.11	0.28	0.89	0.18	5.82	0.18	0.96	0.16	6.00	0.17	1.00	0.17
		30	4.96	0.28	0.82	0.18	5.86	0.19	1.00	0.17	6.00	0.17	1.00	0.17
		50	5.32	0.28	1.00	0.24	5.86	0.18	0.96	0.16	6.00	0.17	1.00	0.17
		70	4.82	0.33	0.89	0.22	5.82	0.18	0.96	0.16	6.00	0.17	1.00	0.17
		100	4.57	0.37	0.89	0.26	5.71	0.19	0.93	0.16	6.00	0.17	1.00	0.17

Continued on next page...

... Table 6 continued.

			60 %				80 %				100 %			
				h_{gc}				h_{gc}				h_{gc}		
Domain	$\|\mathcal{G}\|$	O%	$\|\mathcal{C}\|$	Q	R	M	$\|\mathcal{C}\|$	Q	R	M	$\|\mathcal{C}\|$	Q	R	M
Satellite	6.43	10	1.57	0.71	0.39	0.29	1.82	0.74	0.61	0.42	3.43	0.53	0.79	0.31
		30	1.43	0.69	0.25	0.20	1.43	0.72	0.36	0.34	3.25	0.56	0.79	0.34
		50	1.36	0.81	0.57	0.51	1.86	0.68	0.46	0.34	3.64	0.52	0.86	0.34
		70	1.43	0.78	0.54	0.46	2.07	0.66	0.50	0.34	3.46	0.51	0.75	0.32
		100	1.96	0.83	0.96	0.75	2.07	0.82	1.00	0.67	3.29	0.63	1.00	0.43
Sokoban	7.14	10	2.25	0.60	0.25	0.12	1.75	0.68	0.29	0.14	2.00	0.64	0.29	0.16
		30	1.75	0.70	0.39	0.27	1.54	0.76	0.43	0.31	1.93	0.65	0.29	0.15
		50	1.11	0.87	0.61	0.55	1.46	0.79	0.46	0.36	1.07	0.80	0.36	0.30
		70	1.18	0.87	0.64	0.57	1.21	0.85	0.57	0.54	1.04	0.79	0.29	0.26
		100	1.18	0.95	0.89	0.83	1.07	0.95	0.86	0.84	1.25	0.84	0.54	0.48
Zeno-Travel	6.86	10	1.18	0.75	0.25	0.21	1.43	0.72	0.25	0.19	1.36	0.72	0.21	0.17
		30	1.46	0.73	0.32	0.24	1.11	0.75	0.21	0.21	1.21	0.75	0.25	0.21
		50	1.29	0.84	0.57	0.49	1.14	0.85	0.54	0.54	1.18	0.81	0.39	0.36
		70	1.14	0.92	0.79	0.73	1.11	0.88	0.64	0.63	1.18	0.87	0.64	0.58
		100	1.04	1.00	1.00	0.98	1.00	0.96	0.89	0.89	1.21	0.96	0.96	0.87
ALL	8.29	10	2.38	0.65	0.38	0.19	2.82	0.61	0.42	0.19	3.81	0.53	0.53	0.18
		30	2.21	0.68	0.41	0.24	2.62	0.64	0.41	0.23	3.65	0.55	0.53	0.20
		50	2.24	0.70	0.47	0.32	2.62	0.65	0.47	0.28	3.65	0.57	0.58	0.22
		70	2.18	0.72	0.53	0.38	2.72	0.65	0.49	0.30	3.61	0.57	0.59	0.25
		100	2.09	0.80	0.75	0.59	2.55	0.72	0.71	0.50	3.65	0.62	0.77	0.41

References

1. Amado L, Pereira RF, Aires J, Magnaguagno M, Granada R, Meneguzzi F (2018) Goal recognition in latent space. In: International Joint Conference on Neural Networks, IEEE, IJCNN, pp 1–8, DOI 10.1109/IJCNN.2018.8489653
2. Asai M (2019) Unsupervised grounding of plannable first-order logic representation from images. In: Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, AAAI Press, ICAPS'19, pp 583–591
3. Bisson F, Larochelle H, Kabanza F (2015) Using a recursive neural network to learn an agent's decision model for plan recognition. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, AAAI Press, IJCAI'15, pp 918–924, URL `http://dl.acm.org/citation.cfm?id=2832249.2832376`
4. Chen J, Chen Y, Xu Y, Huang R, Chen Z (2013) A planning approach to the recognition of multiple goals. Int J Intell Syst 28(3):203–216, DOI 10.1002/int.21565
5. Cimatti A, Roveri M (2000) Conformant planning via symbolic model checking. Journal of Artificial Intelligence Research 13:305–338
6. Dolan ED, Moré JJ (2002) Benchmarking optimization software with performance profiles. Math Program 91(2):201–213, DOI 10.1007/s101070100263
7. E-Martin Y, R-Moreno MD, Smith DE (2015) A fast goal recognition technique based on interaction estimates. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, AAAI Press, Buenos, Aires, Argentina, IJCAI'15
8. Fagan M, Cunningham P (2003) Case-based plan recognition in computer games. In: Ashley KD, Bridge DG (eds) International Conference on Case-Based Reasoning Research and Development, Springer, Berlin, Heidelberg, pp 161–170, DOI 10.1007/3-540-45006-8_15
9. Fikes RE, Nilsson NJ (1971) Strips: A new approach to the application of theorem proving to problem solving. Artif Intell 2(3):189 – 208, DOI https://doi.org/10.1016/0004-3702(71)90010-5, URL `http://www.sciencedirect.com/science/article/pii/0004370271900105`
10. Forman G (2003) An extensive empirical study of feature selection metrics for text classification. J Mach Learn Res 3(Mar):1289–1305
11. Freedman RG, Zilberstein S (2017) Integration of planning with recognition for responsive interaction using classical planners. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI Press, AAAI'17, pp 4581–4588, URL `http://dl.acm.org/citation.cfm?id=3298023.3298233`

12. Geffner H, Bonet B (2013) A concise introduction to models and methods for automated planning: Synthesis Lectures on Artificial Intelligence and Machine Learning, 1st edn. Morgan & Claypool Publishers
13. Geib CW, Goldman RP (2001) Plan recognition in intrusion detection systems. In: Proceedings DARPA Information Survivability Conference and Exposition II. DISCEX'01, IEEE, vol 1, pp 46–55, DOI 10.1109/DISCEX.2001.932191
14. Geib CW, Goldman RP (2009) A probabilistic plan recognition algorithm based on plan tree grammars. Artif Intell 173(11):1101 – 1132, DOI 10.1016/j.artint.2009.01.003
15. Ghallab M, Nau D, Traverso P (2004) Automated Planning: theory and practice. San Francisco: Elsevier
16. Ghallab M, Nau D, Traverso P (2016) Automated planning and acting. Cambridge University Press
17. Harman H, Simoens P (2019) Action graphs for performing goal recognition design on human-inhabited environments. Sensors 19(12):2741, DOI 10.3390/s19122741, URL http://dx.doi.org/10.3390/s19122741, special issue on Context-Awareness in the Internet of Things
18. Harman H, Chintamani K, Simoens P (2018) Action trees for scalable goal recognition in robotic applications. In: In Proceedings of the Sixth Workshop on Planning and Robotics, (PlanRob), pp 90–94
19. Helmert M (2006) The fast downward planning system. J Artif Intell Res 26:191–246, DOI 10.1613/jair.1705
20. Helmert M (2009) Concise finite-domain representations for pddl planning tasks. Artif Intell 173(5):503 – 535, DOI https://doi.org/10.1016/j.artint.2008.10.013, URL http://www.sciencedirect.com/science/article/pii/S0004370208001926, special Issue on Advances in Automated Plan Generation
21. Holtzen S, Zhao Y, Gao T, Tenenbaum JB, Zhu S (2016) Inferring human intent from video by sampling hierarchical plans. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, IROS, pp 1489–1496, DOI 10.1109/IROS.2016.7759242
22. Hong J (2001) Goal recognition through goal graph analysis. J Artif Intell Res 15:1–30, DOI 10.1613/jair.830
23. Horvitz E, Breese J, Heckerman D, Hovel D, Rommelse K (1998) The lumière project: Bayesian user modeling for inferring the goals and needs of software users. In: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, UAI'98, pp 256–265, URL http://dl.acm.org/citation.cfm?id=2074094.2074124
24. Hossin M, Sulaiman M (2015) A review on evaluation metrics for data classification evaluations. International Journal of Data Mining & Knowledge Management Process 5(2):1
25. Hu DH, Yang Q (2008) Cigar: Concurrent and interleaving goal and activity recognition. In: Proceedings of the Twenty-Third National Conference on Artificial Intelligence - Volume 3, AAAI Press, AAAI'08, pp 1363–1368, URL http://dl.acm.org/citation.cfm?id=1620270.1620286
26. Jiao P, Xu K, Yue S, Wei X, Sun L (2017) A decentralized partially observable markov decision model with action duration for goal recognition in real time strategy games. Discrete Dyn Nat Soc DOI 10.1155/2017/4580206
27. Kautz HA (1987) A formal theory of plan recognition. PhD thesis, University of Rochester. Department of Computer Science
28. Kautz HA, Allen JF (1986) Generalized plan recognition. In: Proceedings of the Fifth AAAI National Conference on Artificial Intelligence, AAAI Press, AAAI'86, pp 32–37
29. Keren S, Mirsky R, Geib C (2019) Plan activity and intent recognition tutorial. Retrieved from http://www.planrec.org/Tutorial/Resources_files/pair-tutorial.pdf, URL http://www.planrec.org/Tutorial/Resources_files/pair-tutorial.pdf
30. Lemaignan S, Warnier M, Sisbot EA, Clodic A, Alami R (2017) Artificial cognition for social human–robot interaction: An implementation. Artif Intell 247:45 – 69, DOI https://doi.org/10.1016/j.artint.2016.07.002, URL http://www.sciencedirect.com/science/article/pii/S0004370216300790, special Issue on AI and Robotics
31. Levine SJ, Williams BC (2018) Watching and acting together: Concurrent plan recognition and adaptation for human-robot teams. J Artif Intell Res 63:281–359, DOI 10.1613/jair.1.11243
32. Liao L, Fox D, Kautz H (2007) Extracting places and activities from gps traces using hierarchical conditional random fields. Int J Robot Res 26(1):119–134, DOI

10.1177/0278364907073775, URL https://doi.org/10.1177/0278364907073775, https://doi.org/10.1177/0278364907073775
33. Lima WS, Souto E, Rocha T, Pazzi RW, Pramudianto F (2015) User activity recognition for energy saving in smart home environment. In: IEEE Symposium on Computers and Communication (ISCC), IEEE, pp 751–757, DOI 10.1109/ISCC.2015.7405604
34. Masters P, Sardina S (2019) Cost-based goal recognition in navigational domains. J Artif Intell Res 64:197–242, DOI 10.1613/jair.1.11343, URL https://doi.org/10.1613/jair.1.11343
35. McDermott D (2000) The 1998 ai planning system competition. AI Magazine 21(2):35, DOI 10.1609/aimag.v21i2.1506, URL https://search.proquest.com/docview/208131456?accountid=11077
36. Mirsky R, Stern R, Gal K, Kalech M (2018) Sequential plan recognition: An iterative approach to disambiguating between hypotheses. Artif Intell 260:51 – 73, DOI https://doi.org/10.1016/j.artint.2018.03.006, URL http://www.sciencedirect.com/science/article/pii/S0004370218301103
37. Mirsky R, Shalom Y, Majadly A, Gal K, Puzis R, Felner A (2019) New goal recognition algorithms using attack graphs. In: Dolev S, Hendler D, Lodha S, Yung M (eds) Cyber Security Cryptography and Machine Learning, Springer International Publishing, Cham, pp 260–278
38. Palacios H, Geffner H (2009) Compiling uncertainty away in conformant planning problems with bounded width. Journal of Artificial Intelligence Research 35:623–675
39. Pereira RF, Oren N, Meneguzzi F (2017) Landmark-based heuristics for goal recognition. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI Press, AAAI'17, pp 3622–3628, URL http://dl.acm.org/citation.cfm?id=3298023.3298094
40. Pereira RF, Oren N, Meneguzzi F (2019) Landmark-based approaches for goal recognition as planning. arXiv preprint arXiv:190411739
41. Pereira RF, Pereira AG, Meneguzzi F (2019) Landmark-enhanced heuristics for goal recognition in incomplete domain models. In: Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling, AAAI Press, ICAPS'19, pp 329–337
42. Rafferty J, Nugent CD, Liu J, Chen L (2017) From activity recognition to intention recognition for assisted living within smart homes. IEEE T Hum-Mach Syst 47(3):368–379, DOI 10.1109/THMS.2016.2641388
43. Ramírez M, Geffner H (2009) Plan recognition as planning. In: Proceedings of the Twenty-First International Joint Conference on Artifical Intelligence, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, IJCAI'09, pp 1778–1783, URL http://dl.acm.org/citation.cfm?id=1661445.1661731
44. Ramírez M, Geffner H (2010) Probabilistic plan recognition using off-the-shelf classical planners. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI Press, AAAI'10, pp 1121–1126
45. Ramírez M, Geffner H (2011) Goal recognition over pomdps: Inferring the intention of a pomdp agent. In: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, AAAI Press, IJCAI'11, pp 2009–2014, DOI 10.5591/978-1-57735-516-8/IJCAI11-335, URL http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-335
46. Roy PC, Giroux S, Bouchard B, Bouzouane A, Phua C, Tolstikov A, Biswas J (2011) A possibilistic approach for activity recognition in smart homes for cognitive assistance to alzheimer's patients. In: Chen CD Limingand Nugent, Biswas J, Hoey J (eds) Activity Recognition in Pervasive Intelligent Environments, Atlantis Press, Paris, vol 4, pp 33–58, DOI 10.2991/978-94-91216-05-3_2, URL https://doi.org/10.2991/978-94-91216-05-3_2
47. Rubin JE (1967) Set theory for the mathematician. San Francisco (Calif.) : Holden-Day
48. Schmidt C, Sridharan N, Goodson J (1978) The plan recognition problem: An intersection of psychology and artificial intelligence. Artif Intell 11(1):45 – 83, DOI https://doi.org/10.1016/0004-3702(78)90012-7, URL http://www.sciencedirect.com/science/article/pii/0004370278900127, special Issue on Applications to the Sciences and Medicine
49. Shaw PH, Farwer B, Bordini RH (2008) Theoretical and experimental results on the goal-plan tree problem. In: Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, AAMAS'08, vol 3, p 1379–1382

50. Singla G, Cook DJ, Schmitter-Edgecombe M (2010) Recognizing independent and joint activities among multiple residents in smart environments. J Ambient Intell Humaniz Comput 1(1):57–63, DOI 10.1007/s12652-009-0007-1
51. Sohrabi S, Riabov AV, Udrea O (2016) Plan recognition as planning revisited. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, AAAI Press, IJCAI'16, pp 3258–3264, URL http://dl.acm.org/citation.cfm?id=3061053.3061077
52. Thangarajah J, Padgham L, Winikoff M (2003) Detecting & exploiting positive goal interaction in intelligent agents. In: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems, Association for Computing Machinery, New York, NY, USA, AAMAS'03, p 401–408, DOI 10.1145/860575.860640
53. Tremblay S, Fortin-Simard D, Blackburn-Verreault E, Gaboury S, Bouchard B, Bouzouane A (2015) Exploiting environmental sounds for activity recognition in smart homes. In: AAAI Workshop: Artificial Intelligence Applied to Assistive Technologies and Smart Environments, AAAI-ATSE
54. Vattam SS, Aha DW, Floyd M (2014) Case-based plan recognition using action sequence graphs. In: Lamontagne L, Plaza E (eds) Case-Based Reasoning Research and Development, Springer International Publishing, Cham, pp 495–510, DOI 10.1007/978-3-319-11209-1_35
55. Vilain M (1990) Getting serious about parsing plans: A grammatical analysis of plan recognition. In: Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI Press, AAAI'90, pp 190–197, URL http://dl.acm.org/citation.cfm?id=1865499.1865528
56. Wang Z, Boularias A, Mülling K, Schölkopf B, Peters J (2017) Anticipatory action selection for human–robot table tennis. Artif Intell 247:399 – 414, DOI https://doi.org/10.1016/j.artint.2014.11.007, special Issue on AI and Robotics
57. Wu J, Osuntogun A, Choudhury T, Philipose M, Rehg JM (2007) A scalable approach to activity recognition based on object use. In: IEEE Eleventh International Conference on Computer Vision, IEEE, ICCV, pp 1–8, DOI 10.1109/ICCV.2007.4408865
58. Yordanova K, Krüger F, Kirste T (2012) Context aware approach for activity recognition based on precondition-effect rules. In: IEEE International Conference on Pervasive Computing and Communications Workshops, IEEE, PerCom Workshops, pp 602–607, DOI 10.1109/PerComW.2012.6197586
59. Yordanova K, Lüdtke S, Whitehouse S, Krüger F, Paiement A, Mirmehdi M, Craddock I, Kirste T (2019) Analysing cooking behaviour in home settings: Towards health monitoring. Sensors 19(3), DOI 10.3390/s19030646, special Issue on Context-Awareness in the Internet of Things
60. Yue S, Yordanova K, Krüger F, Kirste T, Zha Y (2016) A decentralized partially observable decision model for recognizing the multiagent goal in simulation systems. Discrete Dyn Nat Soc DOI 10.1155/2016/5323121