Journal of Ambient Intelligence and Smart Environments 12 (2020) 79–99
DOI 10.3233/AIS-200556
IOS Press

Action graphs for proactive robot assistance in smart environments

Helen Harman [*] and Pieter Simoens
Department of Information Technology – IDLab, Ghent University – imec, Technologiepark 126, B-9052 Ghent, Belgium
E-mails: Helen.Harman@ugent.be, Pieter.Simoens@ugent.be

Abstract. Smart environments can already observe the actions of a human through pervasive sensors. Based on these observations, our work aims to predict the actions a human is likely to perform next. Predictions can enable a robot to proactively assist humans by autonomously executing an action on their behalf. In this paper, Action Graphs are introduced to model the order constraints between actions. Action Graphs are derived from a problem defined in Planning Domain Definition Language (PDDL). When an action is observed, the node values are updated and next actions predicted. Subsequently, a robot executes one of the predicted actions if it does not impact the flow of the human by obstructing or delaying them. Our Action Graph approach is applied to a kitchen domain.

Keywords: Action prediction, proactive assistance, intention recognition, symbolic AI, smart environment

1. Introduction

An increasing number of robots are being deployed in smart environments to work alongside and assist humans in their daily activities. Rather than a human explicitly stating their goal and conversing with the robot to determine how they will achieve that goal, our work aims to automatically predict what actions the human will perform next (e.g., open a door, switch on an appliance or take an item from a cupboard) and determine which of those actions could be executed by a robot. Figure 1 shows a conceptual overview.

There are many challenges to predicting the actions of humans. For instance, people might pursue multiple goals concurrently, e.g., while waiting for the kettle to boil to make a cup of tea, a person could hang-up their washing. Further, goals are not always well-defined: a prepared cup of tea could be a goal, or a subgoal to be realised as part of the goal to prepare breakfast.

Our proposed system aims to adhere to the following constraints. First, the robot should have minimal impact on the action flow of the observed agent, e.g., not attempt to take an item from a cupboard at the same time as the human. Second, the system should cope with invalid and missing observations caused by noisy erroneous sensor readings and the human making mistakes, e.g., opening the wrong cupboard. Moreover, due to privacy concerns and humans preferring non-intrusiveness sensors [31], some actions, e.g., navigation actions, are unobservable. Third, the algorithms should work online, i.e., update the predictions each time an action is observed.

In this work, actions are discrete and the term observation refers to an action that has already been derived from raw (continuous) sensor data and low-level activity recognition algorithms, such as the work in [34]. The deployment of hardware (i.e., IoT devices and robots) and the integration of activity recognition algorithms are beyond the scope of this paper.

The main contributions of this paper are algorithms for action prediction and for deciding which of the predicted actions should be executed by a robot. The action prediction aspect is performed by an Action Graph, a graphical structure that models the dependencies between the actions an observed (human) agent can perform. An Action Graph is automatically gen-

[*]Corresponding author. E-mail: Helen.Harman@ugent.be.

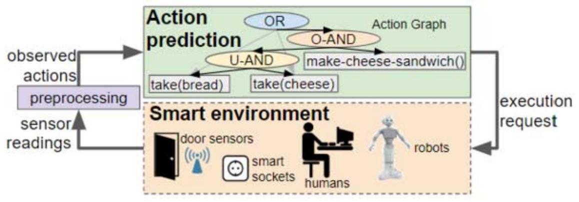


Fig. 1. Conceptual overview.

erated from a problem defined in Planning Domain Definition Language (PDDL). When an observation is received, its node values are updated and the action nodes whose values are above a threshold are extracted, i.e., the predicted actions. The predicted actions are then mapped to a goal state before sending them to a robot. By running a classical task planner, the robot generates a task plan containing a set of actions to reach the goal state from its current state. The robot then decides to pursue the goal based on how short its plan is and how far the human is from performing the predicted action.

We previously presented a short paper on an early version of our work, which focused on single goal recognition [13]. This showed promising results for both the goal recognition times and accuracy of our approach across many different domains. This paper enhances the node value update algorithms; expands the scope of our work by predicting a person's next actions; and adds the robot assistance aspect. Although our work is applicable to different domains (as shown in our previous paper), in this paper a smart kitchen domain is the running example used to illustrate and evaluate the presented algorithms.

The remainder of this paper is structured as follows. A summary of related work is presented in Section 2. In Section 3, the action prediction problem is formalised and the kitchen domain is introduced. Section 4 outlines the structural features of an Action Graph and how it is generated. The node value update rules are detailed in Section 5 and the extraction of the predicted actions in Section 6. How the robot decides whether to execute an action (or not) is presented in Section 7. Section 8 discusses the results of our action prediction experiments and Section 9 presents a proof of concept detailing which actions the robot executes.

2. Related work

Within the literature, studies on recognising the intentions of an observed agent interchange various terms and these often have different definitions, e.g. goal, plan, intention and activity recognition [27,30,34]. In this section some of the existing approaches to intention recognition are introduced, in particular the focus is on those which have similar (hierarchical) structures or take PDDL as input. Intention recognition has been applied to different application domains including computer gaming [7], human-computer interaction [16], energy saving [23] and social robotics [8]. As our work focuses on proactive robotic assistance, other works that integrate intention recognition with robot response are also discussed.

2.1. *Intention recognition*

Methods for intention recognition can be broadly categorised as data-driven and knowledge-driven methods [28,38]. Data-driven approaches train a recognition model from a large dataset [1,3,33,38]. 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 [32,37]. Since our work belongs to the category of knowledge-driven methods, data-driven methods are not discussed further.

Knowledge-driven approaches rely on a logical description of the actions agents can perform. They can be further divided into approaches that search through a library of predefined plans (also known as "recognition as parsing") and approaches that solve a symbolic recognition problem, i.e., "recognition as planning" [20]. These are discussed in turn.

2.1.1. *Recognition as parsing*

Recognition as parsing tends to be fast and allows multiple concurrent plans to be detected [11,19,25], but is often considered to be less flexible [29,30] because a planning library containing all actions and their orderings must be produced a priori. Hierarchical structures are usually developed, which include abstract actions that can be decomposed into concrete (observable) actions.

In [15] a Temporal AND-OR tree was constructed from a library of plans to determine which objects a human will navigate to. Our method of representing a human's actions in an Action Graph is inspired by this AND-OR tree; however, the construction of our graph is considerably different (i.e., Action Graphs are derived from PDDL) and our node value update rules differ. Moreover, their AND-OR tree only contains `ORDERED-AND` nodes, whereas Action Graphs include both `ORDERED` and `UNORDERED-AND` nodes as often actions are not strictly ordered.

A set of action sequence graphs is derived from a library of plans in [35]. 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 they did not investigate multiple interleaving goals.

Kautz et al. [18,19] considered many of the features (e.g., multiple goals, partial plans) we believe are key for goal recognition and action prediction to be deployed within a smart environment. They 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 work describes a formal theory and as far as we are aware was not implemented.

2.1.2. Recognition as planning

Recognition as planning is a more recently proposed approach, in which languages normally associated with task planning, such as PDDL, define the actions agents can perform (along with their preconditions and effects) and world states. This enables a single set of action definitions to be written in a standard language that can be utilised for both action prediction and robot task planning. Moreover, whereas in recognition as parsing there are usually only action definitions, planning-based approaches allow for the inclusion of state knowledge, such as what objects are found within the environment and their locations.

In [29,30] it was proposed to view goal recognition as the inverse of planning. In symbolic task planning, a problem is formulated in terms of an initial and goal state, and task planners find the appropriate set of actions (i.e., a task plan) that changes the current state into the desired goal state [14]. In goal recognition, the initial state, multiple hypothesis goal states, a set of actions and a list of observations are provided as input, and intention recognisers find the most probable goal and/or plan. [30] handles suboptimal plans, but cannot cope with multiple interleaving goals and does not include action prediction. As a planner needs to be called twice for every possible goal, to find the difference in the cost of the plan to reach the goal with and without taking the observations into consideration, this approach is computationally expensive. In [4] the work from [30] was extended to find the joint probability of pairs of goals rather than a single goal. Their work aims to handle multiple interleaving goals but also suffers from large computational costs. Although initial approaches were computationally heavy as they required a task planner to be called multiple times [4,29,30], the latest advances in recognition as planning algorithms have greatly improved this [6,27].

Plan graphs were proposed in [6], to prevent a planner from being called multiple times. 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. [27] significantly reduced the recognition time by finding landmarks, i.e., states that must be passed for a particular goal to be reached. As landmarks rather than all actions are reasoned on, action prediction cannot easily be performed. Moreover, neither [6] nor [27] investigate multiple interleaving plans.

As far as we are aware, our work is the first to take a recognition as planning approach to solve multiple concurrent subgoal action prediction problems, by deriving a graph structure similar to those used by some recognition as parsing approaches from a PDDL defined problem. Moreover, the majority of work on recognition as planning focuses on goal recognition and not on how the observed agent will achieve their aim, e.g., these approaches recognise that a human's goal is to make breakfast but not if they will make tea or coffee to reach that goal, whereas our approach predicts what actions the observed agent will perform next.

2.2. Proactive robot assistance

A robot should assist a human by timely executing an action that helps the human to achieve their goals. In factory and assembly environments the goal and/or plan is often known upfront. Johannsmeier et al. [24] create a high level (team) plan for human and robot workers offline, and map this to robot specific hardware at run-time. Nonetheless, there are many situations in which the plan and goal are not known to the system in advance. Further, several recent studies [2,39] show that humans prefer robots that are proactive, i.e., a robot which autonomously assists a person whenever the robot is able to. Like our approach, the works discussed below process observations to determine part (or potentially all) of the observed agent's plan, which the observer can then execute.

In [21,22] planning networks are derived from causal links extracted from the PDDL problem description. These networks contain the decisions that will be taken by the robot and human (i.e., team plans). They are created offline due to large computational costs, and then processed by an online look-up procedure, which permits a robot to assist a human. Their approach focuses on what uncontrollable decisions a human makes and what action(s) the robot should plan in response. Their work does not aim to prevent the robot from disrupting the flow of a human. Whereas, our work aims to achieve this by only allowing the robot to perform a predicted action if, in comparison, the robot's plan is short and the human is far from completing the prediction.

By contrast, in [9] the most probable goals are determined by running the already cited work on goal recognition as planning [30]. A list of the most likely propositions is then extracted; these propositions are set as the robot's goal. As the goal recogniser [30] calls the planner twice per goal, this approach is computationally expensive. Similar to our work, [21] and [9] take ideas from symbolic task planning as a starting point for their prediction algorithms, but they focus on an agent performing a single goal whereas our work also considers multiple, interleaving and partially completed goals (i.e., subgoals).

3. Problem statement

An Action Graph is created from an action prediction problem defined in PDDL. This section introduces the concepts of PDDL and Domain Transition Graphs (DTGs), an intermediate representation that is processed during creation of an Action Graph. A description of the concrete smart kitchen scenario our work aims to address is also provided.

3.1. Problem formalisation

Formally, an intention recognition problem can be defined as $T = (F, I, A, O, \mathcal{G})$, where F is a set of atoms, $I \subseteq F$ is the initial state, A is a set of actions, $\mathcal{G}$ is the set of all possible goals and O is the sequence of observed actions [30]. Actions contain preconditions $a_{\text{pre}} \subseteq F$ and effects a_{eff}, which includes add effects $a_{\text{eff}+} \subseteq F$, e.g., `(open cupboard)`, and delete effects $a_{\text{eff}-} \subseteq F$, e.g., `(not (open cupboard))`. Our action prediction method produces $\mathcal{A}$, which contains the actions the observee is likely to perform next.

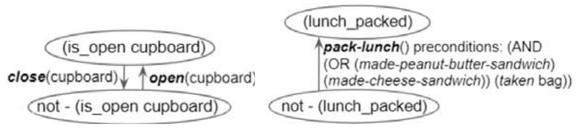
Intention recognition is often viewed as the inverse of planning. When invoked, planners translate PDDL into structures, e.g., DTGs, that can be efficiently searched [14,17]. DTGs are created from a planning problem, i.e., $P = (F, I, A, G)$ [10,12]. G is a goal state, specified in PDDL using *or* and *and* statements.

Each variable has its own DTG in which the nodes contain a variable's possible values. The edges are called transitions and describe what actions (and/or axioms) are required to move between values. If there are multiple ways (plans/actions) to transition between two values, the corresponding edge will have multiple labels.

3.2. Running example: Smart home kitchen

Throughout this paper, examples from the *kitchen* dataset are provided to help describe our approach. This dataset was created by Ramırez et al. [29,30] based on the work by Wu et al. [36], but has been extended to include `open(?container)`, `close(?container)` and `take(?item ?container)` actions. In this dataset there are 3 hypothesis goals: $\mathcal{G} = \{$`(made_breakfast)`, `(lunch_packed)` and `(made_dinner)`$\}$. These goals require multiple items to be taken and other actions to be performed, e.g., part of the plan to make breakfast involves taking bread and making toast. Each goal can be achieved by multiple plans, e.g., for `(lunch_packed)` to be reached a person must always perform the `take(lunch_bag)` action and either perform the `make-peanut-butter-sandwich()` or `make-cheese-sandwich()` action.

A subset of the DTGs created from a kitchen problem are depicted in Fig. 2. In this problem, variables transition between true and false values. As making a packed lunch has multiple possible plans, the transition to the value `(lunch_packed)` being true has



(a) Open cupboard. (b) Pack lunch.

Fig. 2. Example DTGs taken from a kitchen problem.

(at least) two transition labels. Details on the plans producible from this domain are provided in Appendix A.

4. Action graphs

Action prediction is performed by building an Action Graph, updating the value of the graph's nodes when an observation (i.e., action) is received, and then extracting the highest valued actions and their dependencies. This section provides details on the structural features of an Action Graph and on how it is derived from a PDDL defined problem.

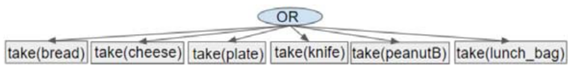
4.1. Structural features

Action Graphs model the order constraints, i.e., *dependencies*, between actions. Dependencies are defined, in this paper, as actions that set one or more of the dependant's preconditions. For instance, action 1 is said to be dependent on action 2 if action 2 fulfils one (or more) of action 1's preconditions. An Action Graph contains `OR`, `ORDERED-AND`, `UNORDERED-AND` and leaf nodes. Leaf nodes are also referred to as action nodes, as each one is associated with an action. `ORDERED-AND` denotes that its children are performed in order, and `UNORDERED-AND` denotes its children can be performed in any order. For `OR` nodes, one or more of its children can be performed. The root node is always an `OR` node, and will receive a new child for every action inserted into the graph. Throughout the paper, unless otherwise stated, the term parent(s) always refers to the direct parent(s) of a node, the same goes for child/children.

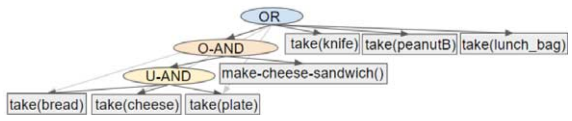
The term graph is used, rather than tree, because action and `ORDERED-AND` nodes can have multiple parent nodes. Action Graphs are acyclic, i.e., do not contain any cycles; actions have a fixed set of dependencies, and if an action 1 depends on action 2, action 2 cannot depend on action 1.

4.2. Creation

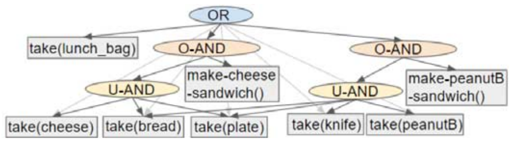
DTGs [14] are generated from a planning problem $P = (F, I, A, G)$ and not from an action prediction problem $T = (F, I, A, O, \mathcal{G})$. Therefore, a goal state G is created by placing all elements in the set of hypothesis goals ($\mathcal{G}$) in an `or` statement. For each DTG, the DTG's transition labels are iterated over to extract actions along with their dependencies, and insert them into the Action Graph. The Action Graph construction



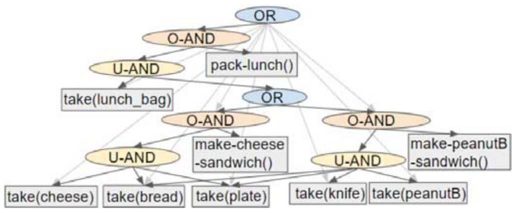
(a) Actions without any dependencies are appended to the root node's children.



(b) The `make-cheese-sandwich()` action has three dependencies which are set as an `U-AND` node's children. The `U-AND` node followed by the `make-cheese-sandwich()` action node are set as an `O-AND` node's children.



(c) The `make-peanutB-sandwich()` action is inserted in the same way as the `make-cheese-sandwich()` action.



(d) The `pack-lunch` action has a dependency on `take(lunch_bag)` and either the `make-peanutB-sandwich()` or `make-cheese-sandwich()` actions. Therefore, as well as inserting a new `U-AND` and `O-AND`, an `OR` node is also created.

Fig. 3. Figures showing some of the steps performed during the creation of the Action Graph for the kitchen domain. To simplify our examples `open` and `close` actions are excluded, i.e., the original unmodified kitchen problem [30] is used. For readability, arrows from the root node to nodes that have another parent have been put in light grey. O-AND stands for `ORDERED-AND` and U-AND is `UNORDERED-AND`. Some action names have been shortened, e.g., `peanut-butter` to `peanutB`.

steps described in this section are illustrated in Fig. 3. For simplicity, open/close actions are omitted.

Actions associated with transition labels that do not have any preconditions, and therefore are assumed to have no dependencies, are appended as children of the root node, e.g., `take(bread)` shown in Fig. 3(a). These nodes may gain additional parent(s) when subsequent actions, which they are dependencies of, are inserted into the graph.

Actions associated with transition labels with preconditions are inserted after all of their dependencies. To perform this insertion several nodes are created including: the action node itself; if the action has more than one dependency an UNORDERED-AND node containing all the dependencies as its children; and an ORDERED-AND node whose children are the UNORDERED-AND node (or single dependency) followed by the action node. The ORDERED-AND node is appended as a child of the root node. Figures 3(b) and 3(c) show the Action Graphs after this process has been performed on the make-cheese-sandwich() and make-peanut-butter-sandwich() actions, respectively.

If an action has dependencies, it will only ever have one parent, of type ORDERED-AND. Therefore, when an action has dependencies, its parent ORDERED-AND node is used to connect it to its dependants.

For transitions with multiple labels (e.g., the DTG in Fig. 2(b)), an OR node is inserted into the Action Graph. This node is appended to the (UN)ORDERED-AND node's children and the multiple possible dependencies set as the OR node's children. An example is provided in Fig. 3(d) in which the pack-lunch() action requires either make-peanut-butter-sandwich() or make-cheese-sandwich() to be performed first.

Each action node may optionally have a list of *reverse actions*. Two actions are the reverse of each other if one sets an atom $a_{\text{eff}+}$ and the other deletes it $a_{\text{eff}-}$. For example, close(cupboard) is the reverse of open(cupboard). Reverse actions are identified by checking if a DTG has opposite edges between the same two values, e.g., the DTG depicted in Fig. 2(a). Details on the importance of reverse actions are provided in Section 5.2.

The Action Graph generation process finishes when all actions have been inserted and reverse actions have been identified. Action Graphs are quick to generate (see Experiment in Section 8) but the process will typically be performed offline. If new actions become available, e.g., because the human installs a new IoT device in their house, these new actions can be inserted into the graph by executing the method described above. The size of the produced graph depends on the number of actions and what constraints those actions have. Readers familiar with FD [14] will know that DTGs also contain transitions labelled with axioms; axioms are not inserted into the Action Graph.

5. Node value updates

Every time the environment observes the human performing an action, the Action Graph's node values are updated according to the algorithms described in this section. In this section, the term AND node refers to either a UNORDERED-AND or a ORDERED-AND node.

5.1. Updating node values based on observations

Each node has an initial (minimum) value of 0 and a maximum value of 1. When an observation $o \in O$ is received, the value of the corresponding action node is set to 1. Subsequently, a value update procedure is executed, which contains an upward and a downward pass. The upward pass traverses the Action Graph from the observed action's node to the root node. Then the downward pass updates all node values in a depth first trajectory starting at the root. The purpose of this value update process is to increase the value of the actions, which are most likely to be performed next.

Pseudo-code of the upward pass is shown in Algorithm 1. After setting the value of the observed action's node to 1 (line 3), the parent nodes are updated recursively (lines 13). If the observed action has a reverse action, the reverse action's value is reset (see Section 5.2). It does not matter in which order a node's parents are updated. During the upwards recursion, the argument of the method call in line 1 will always be of node type OR, UNORDERED-AND or ORDERED-AND.

- If the argument is a node of type OR, its value is set to the maximum value of its children (line 6) because, by the nature of the node itself, its value should not depend on the number of children it has.
- If the argument is a node of type UNORDERED-AND, its value becomes the mean of its children's values. We considered calculating product, rather than mean, but with product the size of the subtrees has a much larger effect on the probability of a goal, i.e., strongly favours shorter plans. To calculate the mean, a value of 0 replaces the actual value of unobserved action nodes (see lines 15–18). If the algorithm were to use the actual node value, node values would be increased too rapidly which causes a high rate of false positive predictions on subgoal action prediction problems.
- If the argument is a node of type ORDERED-AND, its value becomes the mean value of its children which come after (and including) the last child whose value is 1. Like UNORDERED-AND, an ORDERED-AND node's value is based on how completed its children are. Moreover, due to

Algorithm 1 Update node value upwards

```
1:  function UPDATE_NODE_VALUE_UPWARDS(node)
2:      if node is an action node then                                      ▷ The observed action
3:          $node.value = 1.0$
4:          for each r in node.reverseActions do r.RESET_NODE_VALUE() end for    ▷ See Section 5.2
5:      else if node is an OR node then
6:          $node.value = \max(children)$                          ▷ Maximum value of node's children
7:      else if node is an UNORDERED-AND node then
8:          $node.value = \overline{v(c)} \quad \forall c \in node.children$     ▷ Mean of node's children
9:      else if node is an ORDERED-AND node then
10:          $U \leftarrow c \quad \forall c \in node.children$ after last child with node value 1
11:          $node.value = \overline{v(c)} \quad \forall c \in U$   ▷ Mean of children after (and including) 1st observed child
12:     end if
13:     for each parent in node.parents do UPDATE_NODE_VALUE_UPWARDS(parent) end for
14: end function
15: function v(node)
16:     if node is an action node and $node.value < 1.0$ then return 0.0
17:     else return node.value end if
18: end function
```

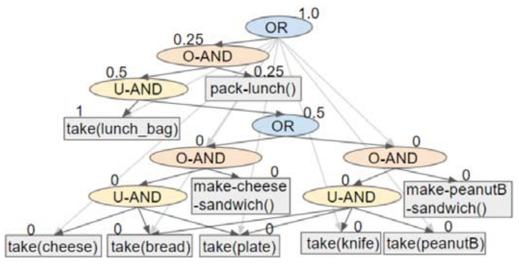
Algorithm 2 Update node value downwards (depth-first)

```
1: function UPDATE_NODE_VALUE_DOWNWARDS(node)
2:     if node is an ORDERED-AND node or node is an UNORDERED-AND node then
3:         for each child in node.children do $child.value = \max(child.value, node.value)$ end for
4:     end if
5:     for each child in node.children do UPDATE_NODE_VALUE_DOWNWARDS(child) end for
6: end function
```

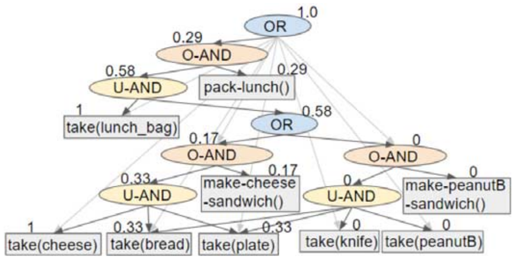
the order constraints imposed on ORDERED-AND nodes' children, we can assume that the left most branch has been completed if the right branch has been observed. This improves Action Graphs' ability to handle missing observations.

After Algorithm 1 returns, a downward pass, shown in Algorithm 2, is performed. Starting from the root node, this recurs over all nodes in a depth-first manner. Direct children of AND nodes are assigned the value of their parent if that value is larger. The direct children of OR nodes are not updated.

To demonstrate how these algorithms affect the nodes' values, an example is provided and depicted in Fig. 4. When processing the first observation, namely, `take(lunch_bag)`, the only action node whose value is affected is `pack-lunch()` as it is connected to `take(lunch_bag)` via AND nodes. At this point determining if the person is making a cheese or peanut butter sandwich is impossible. When `take(cheese)` is observed, the values of the `pack-lunch()` action and all the actions associated



(a) Node values after `take(lunch_bag)` has been observed



(b) Node values after `take(cheese)` has been observed.

Fig. 4. Figures showing the node values after they have been updated by Algorithms 1 and 2.

with making a cheese sandwich are increased. `make-cheese-sandwich()` node's value is increased by 0.17, whereas `pack-lunch()` node's value is increased by 0.04. This is because the further (i.e., number of nodes that must be traversed) an action is from the observed action the less its value is increased (as the human could be aiming to achieve a subgoal).

5.2. *Node value updates with reverse actions*

If the state set by an observed action is reversed, the nodes whose values were increased when the action was observed should be decreased. For example, closing a cupboard is the reverse of opening that cupboard and reduces the chance of an item being taken from the cupboard, and thus any action which requires those items. Moreover, when a node value is set to 1 the action prediction algorithms will not consider this action as a plausible candidate for future actions. This is not a valid assumption in real-world domains, where people can, e.g., open a cupboard multiple times. When an action is observed, the first step (i.e., line 4 of Algorithm 1) resets the value of the reverse action nodes. If the node being reset is the child of an `AND` node, all its non-observed siblings must also be reset to zero. Pseudo-code and an example is provided in Appendix B.

After setting the reverse action node's value to 0, each of its parents are iterated over. If the parent is an `AND` node, its children that have not yet been observed (and have not already been reset) also need to be reset. An action node is simply reset by setting its value to 0. For children that are `OR` nodes, their value becomes the maximum of their children because during the downwards pass (Algorithm 2) their value was updated but their children's values were not. For `AND` nodes the children are recursively reset, as they were also updated when the reverse action was observed. Once all of its children have been reset, the `AND` node's value is then updated to the mean value of its children, and the algorithm continues to recurs upwards until an `OR` node, e.g., the root, is reached.

6. Action prediction with action graphs

Action prediction is performed by finding action nodes with a value greater than threshold θ and extracting their dependencies, including the dependencies' dependencies. Dependencies are extracted because an action can only be performed if its dependencies are executed first. This process results in a map $\mathcal{A}$, which maps each action node with a value greater than θ to a list of its dependencies.

Dependencies are extracted by traversing depth-first starting from an `ORDERED-AND` node (i.e., the parent of a node with dependencies), and appending all actions nodes encountered to the dependency list. For `OR` nodes, only the most likely child is traversed, and for `UNORDERED-AND` nodes, the children are first sorted, highest value first. Note, the action (key) itself is also appended to the end of the dependency list during this depth-first traversal.

If an action node (a) in the map's keys is in the dependency list of another action in the map, then the latter dependency list will contain all elements in the dependency list of a. Therefore, the entry with key a is removed from the map ($\mathcal{A}$). This reduces the amount of unnecessary data; however, it is still possible (and very likely) for some actions to appear in multiple dependency lists.

Actions which are attached to an `OR` node (not including the root) that have a lower value than another action attached to the same `OR` node are removed from the predicted actions. This is because `OR` branches often contain similar actions, and thus when one branch is above the threshold so are the others, e.g., making a cup of coffee and making a cup of tea require many similar actions. The map of predicted actions is recreated every time an observation is received, thus removed actions can reappear in the list during subsequent iterations.

7. Proactive robot assistance

A robot can proactively assist a human by executing one of the predicted actions. The decision about which action to execute is based on the value of the predicted action and on how long it would take the robot or human to execute that action. An overview of this process is depicted in Fig. 5. This section introduces which predicted action is selected, what information is sent to the robot and the robot's decision making process.

The Action Predictor checks if a robot can execute one of the predicted actions ($\mathcal{A}$). Starting with actions in the highest valued action's (key's) dependency list, each action is checked to see if it appears in a predefined list of actions the robot is capable of executing. If it does, an execution request is sent to the robot and if the robot chooses to execute that request, the iteration finishes. Otherwise, the iteration continues un-

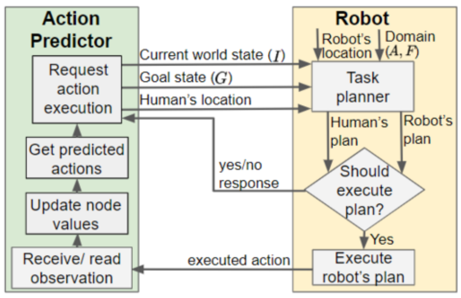


Fig. 5. Overview of the processes the action predictor and robot perform.

til all predicted actions (and their dependencies) have been processed.

The robot decides to execute the request by generating task plans for itself and the human with the task planner of [26]. As introduced in Section 3, this planner solves a planning problem $P = (F, I, A, G)$. The world state I is provided by the Action Predictor, which maintains an updated world model by applying observed actions to this model. The predicted action is converted to a goal state G, which is determined from the action's effects (a_{eff}). When the robot receives a request it will call the planner twice, once from the perspective of itself and once from the perspective of the human (i.e., as if the robot is at the human's location in the world model I). The length of the plans, i.e., number of actions in the plans, produced by the task planner are provided to Eq. (1).

The robot decides to execute the request by comparing the length of its own plan with the human's plan, see Eq. (1). In this equation l_r is the length of the robot's plan, l_h is the length of the human's plan, w is a weight factor ($0 \leqslant w \leqslant 1$) and σ is the sigmoid function that maps the plan length to a value between 0 and 1. The length of the robot's plan is converted to a negative value, so that the longer the robot's plan the lower the result (i.e., inverse sigmoid). If the result of Eq. (1) is above threshold β, the robot will execute its plan.

$$\gamma(w) = \big(\sigma(-l_r) \times w\big) + \big(\sigma(l_h) \times (1 - w)\big) \quad (1)$$

This decision rule is designed to help prevent the robot from obstructing or delaying the human. For example, if the human is about to take an item from a cupboard (i.e., l_h is small), the robot should not obstruct them by attempting to take an item from the same cupboard. We also assume the human is friendly towards the robot, i.e., the robot will announce its plan and the human will not perform the actions within the robot's plan. Thus, the robot should not force the human to wait for it to execute a long plan.

Actions that are absent from the Action Graph, as they are not defined in the action prediction problem the graph was generated from, could still be executed by the robot. This is because the robot's own set of actions, defined in PDDL, could include additional (unobservable) actions.

8. Experiments: Action prediction accuracy

The experiments in this section aim to show how accurately Action Graphs predict a person's actions. This section describes the setup, then reports the action prediction results. The action prediction results include the accuracy for when a simulated human is pursuing one of the hypothesis goals, a subgoal and multiple (interleaving) subgoals.

8.1. Setup

Our Action Graph based method for action prediction was tested on the kitchen domain, introduced in Section 3.2. When running the task planner [26] to create list of observations (i.e., actions) and the ground truth, a `move(?s ?g)` action definition was also included; in the original dataset spatial layout was not accounted for. This action enabled rational plans to be generated, i.e., the simulated human takes all items from the closest location before moving to the next location, instead of re-visiting the same location multiple times. The layout of the modelled environment is depicted in Fig. 6. Unless otherwise stated, for

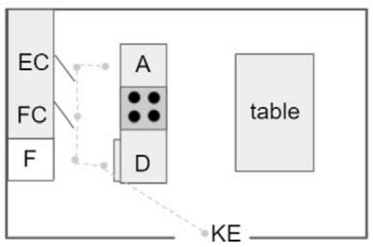


Fig. 6. The layout of the environment used for the simulated experiments. EC is equipment cupboard, FC food cupboard, F fridge, D draw, A appliance (e.g., toaster and kettle) and KE is kitchen entrance. There are waypoints in front of containers and at the entrance to the kitchen, the dashed lines indicate how the locations are linked for the planning of move actions. This environment is based on our HomeLab[1].

[1]https://www.imec-int.com/en/homelab.

each experiment the human starts from the kitchen entrance.

Statements, e.g., `(not(open fridge))`, were appended to the planner's goal to force all containers to be closed before the simulation finished, without these statements the task plans would not have contained `close(?container)` actions. All actions in this domain, except move actions, are assumed to be observable by mapping sensor readings to observations or by applying activity recognition algorithms such as [34].

The Action Graph creation for the kitchen problem, including transforming the PDDL to DTGs, only takes an average of 0.13 seconds on a virtual machine with 8 GB RAM and 2 CPUs (2.90 GHz). Having said that, Action Graphs can be created offline; the algorithms for traversing the graph to update the node values and find the predicted actions must be performed online. Our DTGs to Action Graph transformation method has a time complexity of $O(n^2)$ as for each action (not applicable to the initial state) all actions are iterated over to find the actions' dependencies. Algorithms 1, 2 and 3 have a linear (worst-case) time complexity, i.e., $O(n)$ with respect to the number of actions, as these algorithms traverse the graph (which has a tree-like structure) without traversing the same edge twice.

The results, in this section, report the number of correctly predicted (i.e., true positives) and incorrectly predicted (i.e., false positives) actions after observing the first 10, 30, 50, and 70% of the actions in the plan produced by the task planner. Already observed actions are not included in the comparison, and move actions are not accounted for since they are assumed unobservable. Experiments were repeated for various values of θ, which is the minimum value of an action node to be included in the list of predicted actions.

8.2. *Hypothesis goal action prediction*

In the kitchen dataset there are three hypothesis goals: $\mathcal{G} = \{$ `(made_breakfast)`, `(lunch_packed)`, `(made_dinner)`$\}$. There are multiple options a human could choose to achieve each of these goals:

- `(made_breakfast)` by making coffee or tea;
- `(made_dinner)` by either making a salad or a cheese sandwich, or both;
- `(lunch_packed)` by either making a peanut butter or cheese sandwich.

To produce the plans of these 7 options, the goal provided to the task planner was set to an `and` statement, e.g., `(and ( (made_coffee) (made_breakfast)))`. The planner [26] is not guaranteed to find the optimal plan; thus, we also inserted `not` statements into the goal to prevent the human taking additional items, e.g., to prevent the human taking a tea bag when they are making coffee. The results for action prediction when the simulated agent is performing each one of these options are presented in Table 1.

Both lower values of θ and higher numbers of observations ($|O|$) resulted in more predictions being made. At $\theta = 0.65$ more true positives were returned in comparison to higher values of θ, but also a higher number of false predictions were made. On average the number of false positives reduced more than the number of true positives as θ is increased. This shows that at least some of the true positive actions have a higher value than those that were falsely identified.

We considered increasing the value of θ as more observations are observed but this would have little effect on which actions the robot executes. The list of predicted actions is sorted (highest value first) prior to checking if the robot can execute each action in turn; therefore, the robot is likely to execute actions correctly even when false positives are included within these predictions. Moreover, in all presented experiments no predictions were made when 10% of observations are provided. When such a small number of observations are provided, it is unclear what the human's intentions are; thus, a robot should not start acting.

Even with a high threshold value $\theta = 0.85$, the number of false positives in relation to the plan length $|a|$ remains high for three options: 1) `(lunch_packed)` including `(made_cheese_sandwich)`, 2) `(made_dinner)` including `(made_cheese_sandwich)` and 3) `(made_dinner)` including `(made_cheese_sandwich)` and `(made_salad)`. The main reason is that the plans to make a peanut butter sandwich and a cheese sandwich are very similar; therefore, the predicted actions are the union of actions required to make both types of sandwich.

8.3. *Single subgoal action prediction*

To demonstrate how well our approach predicts actions of subgoals, plans (ground truth) for 6 subgoals were created using the method described in Section 8.1, and the accuracy of the predictions was recorded after 10, 30, 50 and 70% of actions had been observed. No modifications were made to the list

Table 1

Number of true positives (TP) and false positives (FP) for different goals (and plans) when the first 10, 30, 50 and 70% of actions have been observed. $|a|$ is the total number of observable actions in the plan, and $|O|$ the number of processed observations

Goal	$\|a\|$	$\|O\|$	0.65		0.75		0.85		0.95	
			TP	FP	TP	FP	TP	FP	TP	FP
`(made_breakfast)` including: `(made_coffee)`	28	3	0	0	0	0	0	0	0	0
		8	7	1	4	0	4	0	0	0
		14	11	3	9	3	9	2	4	0
		20	6	6	6	5	4	4	0	2
`(made_breakfast)` including: `(made_tea)`	27	3	0	0	0	0	0	0	0	0
		8	7	1	4	0	4	0	0	0
		14	10	7	5	4	5	3	1	2
		19	6	7	6	6	4	4	1	2
`(lunch_packed)` including: `(made_peanut_butter_sandwich)`	15	2	0	0	0	0	0	0	0	0
		5	4	3	4	0	0	0	0	0
		8	2	5	2	1	2	1	2	0
		11	2	10	0	3	0	1	0	1
`(lunch_packed)` including: `(made_cheese_sandwich)`	12	1	0	0	0	0	0	0	0	0
		4	3	0	0	0	0	0	0	0
		6	2	3	2	0	2	0	1	0
		8	2	7	0	5	0	3	0	0
`(made_dinner)` including: `(made_cheese_sandwich)` and `(made_salad)`	14	1	0	0	0	0	0	0	0	0
		4	3	0	0	0	0	0	0	0
		7	3	6	1	3	1	0	1	0
		10	2	6	2	4	0	4	0	0
`(made_dinner)` including: `(made_cheese_sandwich)`	11	1	0	0	0	0	0	0	0	0
		3	0	0	0	0	0	0	0	0
		6	2	3	2	0	2	0	0	0
		8	1	8	0	5	0	3	0	0
`(made_dinner)` including: `(made_salad)`	6	1	0	0	0	0	0	0	0	0
		2	2	0	0	0	0	0	0	0
		3	1	0	1	0	1	0	0	0
		4	1	0	0	0	0	0	0	0

of hypothesis goals, which is processed to transform PDDL into DTGs.

The results, presented in Table 2, show a similar trend as the results in the previous section. When (non-distinctive) actions belonging to multiple goals are observed, often actions belong to both goals will appear in the predicted actions of a single goal. For example, the false positives returned for `(made_buttered_toast)` are contained within the plan to `(made_peanut_butter_sandwich)`, both these goals have similar actions, i.e., require the human to retrieve bread, retrieve a knife and open the fridge.

In the case of `(made_coffee)`, the 2 false positives (for $\theta = 0.85$), and all actions required to make coffee, are contained within the plan to reach `(made_breakfast)`. The more observed a plan is the more likely the human is to be aiming to achieve its goal, thus the more likely they are to perform other actions in that plan. When θ is increased to 0.95, all the predicted actions are correct as those actions further away (in the Action Graph structure) from the observed actions have a lower value.

8.4. Multiple interleaving subgoals

In this section, results are presented for when a simulated human is interleaving actions to complete 2 subgoals. Subgoals were chosen, rather than the original

Table 2

Action prediction for the subgoals, found in the kitchen domain

Goal	$\|a\|$	$\|O\|$	0.75		0.85		0.95	
			TP	FP	TP	FP	TP	FP
`(made_coffee)`	15	2	0	0	0	0	0	0
		5	7	0	0	0	0	0
		8	5	2	5	2	2	0
		11	3	3	3	2	3	0
`(made_peanut_butter_sandwich)`	13	1	0	0	0	0	0	0
		4	0	0	0	0	0	0
		7	3	1	3	0	0	0
		9	2	1	2	1	2	0
`(made_buttered_toast)`	12	1	0	0	0	0	0	0
		4	0	0	0	0	0	0
		6	4	0	4	0	0	0
		8	2	4	2	3	0	0
`(made_tea)`	11	1	0	0	0	0	0	0
		3	0	0	0	0	0	0
		6	3	0	3	0	0	0
		8	2	0	2	0	0	0
`(made_cheese_sandwich)`	10	1	0	0	0	0	0	0
		3	0	0	0	0	0	0
		5	2	0	2	0	0	0
		7	1	3	1	0	1	0
`(made_salad)`	6	1	0	0	0	0	0	0
		2	0	0	0	0	0	0
		3	1	0	1	0	0	0
		4	0	0	0	0	0	0

hypothesis goals, as there are a higher number of subgoals, allowing more combinations to be tested. Moreover, if multiple of hypothesis goals were pursued, nearly all actions would be required, making it trivial to correctly predict actions.

When a human will switch between plans is non-deterministic; therefore, the results for 2 different methods of selecting when to switch between the plans are provided. In the first approach, a single call to the planner was performed with the goal set using an `and` statement, e.g., `(and (made_coffee) (made_salad))`.

For the other approach the plans for both subgoals were created independently, then interleaved based on the probability (p) of a human switching between the two plans. If this probability is set to 1, one action from each of the plans will be selected in turn (i.e., alternating); if one plan is longer, any remaining actions are appended to the end. Actions are not repeated: if an action has already been executed because it was part of the other plan, the subsequent action is selected. The results show the average of 5 runs for when the probability of switching plans was set to 1, 0.75, 0.5, 0.25 and 0. As before, the predictions for when 10, 30, 50 and 70% of the actions have been observed are compared to the real (interleaving) plan. The average results is shown in Table 3 and results for each individual pair of subgoals are provided in Appendix C.

There is a large amount of variation in the results for the different ways of interleaving two goals. In general, when 50% and 70% of actions had been observed, the number of true positive predictions decreases when the probability of switching plans (p) is reduced. With lower switching probabilities, most of the observations are from only one of the goals, hence the actions from the other goal cannot be predicted. For 30% of observations, the number of true positives often decreases when p is increased as fewer actions from a single goal

Table 3

Action prediction for combinations of 2 plans that achieve subgoals, found in the kitchen domain. θ is set to 0.85

Goal	$\|a\|$	$\|O\|$	Planner		$p = 1$		$p = 0.75$		$p = 0.5$		$p = 0.25$		$p = 0$	
			TP	FP	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
Average	17.93	1.93	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
		5.47	0.80	0.00	0.67	0.07	0.57	0.07	1.10	0.17	1.32	0.25	2.07	0.40
		9.33	2.27	1.07	4.13	0.73	3.49	0.84	3.38	0.97	2.28	1.04	1.00	1.40
		12.60	2.40	2.60	2.73	2.00	2.69	2.08	2.57	2.09	2.30	2.32	1.07	1.40

are observed, making it more challenging to make any predictions.

9. Experiments: Robotic assistance

In this section, how the robot decides which of the predicted actions it can execute on behalf of the human is evaluated. The effect changing the values of β, w and θ, and of receiving invalid observations, has on which actions the robot executes is discussed.

9.1. Setup

The simulated robot is capable of all `take (?item)` and `take(?item ?container)` actions. How many actions were (correctly) taken over by the robot is discussed, as well as if the human was temporarily blocked from performing an action, e.g., because they had to wait for the robot to take an item.

The simulated human's plan may be affected by the robot. When the robot changes the state of the environment and a precondition of the human's next action is no longer met, a new plan is produced for the human. It can also happen that the next action the human wants to execute is in the robot's current plan. If so, this action is skipped and the human attempts to proceed with the next action in its plan, without replanning.

Simulations were ran for the three hypothesis goals: to make breakfast, dinner and lunch. The resulting action sequence for making breakfast is shown in Fig. 7. For making dinner and lunch, the action sequences are provided in Appendix D. To produce these results, θ was set to 0.85, β to 0.45 and w to 0.5.

9.2. Results and discussion

To make breakfast, the human requires 32 actions without robotic assistance and 29 actions with robotic assistance. The robot correctly opened the food cupboard, took the bread and took a cloth. The robot did not perform any incorrect actions, i.e., no actions were executed that were not in the human's original plan. During this experiment the human waited for the robot to open the food cupboard as it took a while for the robot, i.e., 3 move actions, to reach its location. An impatient human could have opened the cupboard themselves enabling the robot to then just take the bread, or continued with actions later in their plan. The human's wait times were more prominent when the robot assisted the human to make dinner and pack lunch. Because the plans to reach these two goals are short, the human could not perform any further actions.

9.2.1. Changing β

The threshold β controls if the robot should execute the request or not based on the length of its plan and the plan it generates for the human to perform that request, see Eq. (1). When β is increased to 0.5, the robot does not execute any actions, unless its starting location is adjusted in order to shorten its plan. When the human was making breakfast and the robot started near the fridge, the robot correctly retrieved bread, but then incorrectly retrieved sugar. When making breakfast the human has the option of making a coffee with sugar, and as the actions to make tea and coffee are similar, the actions in the plan to make coffee also had high values. For making lunch and dinner, when the robot is placed near the cupboard, the robot correctly opened the equipment cupboard and took a plate.

When β was decreased to 0.4 and the human was making breakfast (with the original initial locations), the robot retrieved the bread and a bowl and incorrectly retrieved sugar. Prior to the robot starting to get a bowl, the human had opened the equipment cupboard in which the bowl is stored, and both the human and robot took items from the equipment cupboard simultaneously. In a real-world setting, such collisions must be prevented, e.g., by setting β to a higher value. For making dinner and making lunch, lowering the value of β had no effect on which actions the robot executed.

9.2.2. Changing w

By changing w, the robot's decision can be either weighted towards the robot's or human's plans' length.

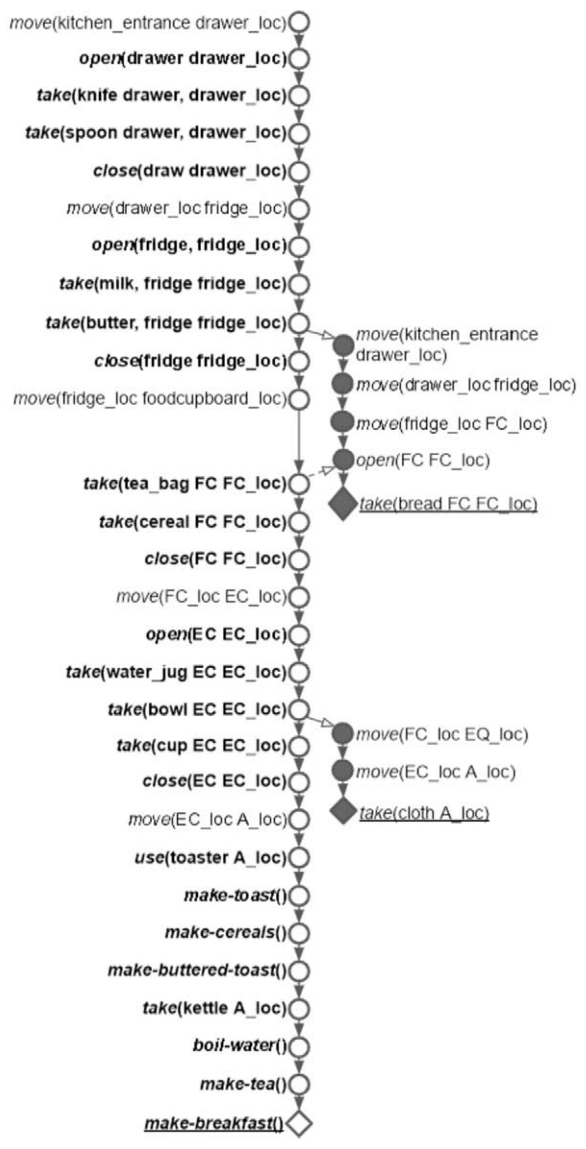


Fig. 7. The human action sequence when their goal is to make breakfast (left) and the actions executed by the robot (right) when its initial location is `kitchen_entrance`. θ is set to 0.85, $\beta = 0.45$ and $w = 0.5$. Actions in bold are observable. Thick arrows pointing right indicate when the robot started execution, and dashed arrows when the human waited for the robot before executing an action.

When weighted towards the robot's plan (i.e., $w = 0.6$), the robot did not execute any actions for any of the goals due to how long its initial plan was. Lowering w had the same effect as decreasing β.

9.2.3. *Changing θ*

Only action nodes with a value above the threshold θ are returned by the action predictor and are thus possible candidates for being executed by the robot. For making breakfast, when θ was decreased to 0.7 the robot correctly executed the following actions: `open(foodcupboard)`, `take(bread foodcupboard)`, `open(equipmentcupboard)`, `take(bowl equipmentcupboard)` and `take (cloth)`. This resulted in the human only performing 27 actions, instead of 32 without robotic assistance. When θ was reduced further, the robot executed incorrect actions, i.e., false positive predictions, as it started acting before enough observations to determine the human's true intentions had been received. At $\theta = 0.6$, the robot executed 2 incorrect actions (i.e., retrieved peanut butter and sugar), and at $\theta \leqslant 0.4$ three incorrect actions were executed. Increasing θ reduces the number of actions executed by the robot and at $\theta = 0.9$ the robot did not execute any actions.

At $\theta \geqslant 0.25$ and $\theta \leqslant 0.85$, and when the human's goal was to make dinner or pack lunch, the robot correctly opened the equipment cupboard and took the plate for the human. When θ was lowered to 0.2, the robot also correctly opened the food cupboard and took the bread. Lowering θ further resulted in the robot performing incorrect actions as the robot acted too soon.

9.2.4. *Noisy observations*

As sensors may provide erroneous readings and humans can make mistakes (e.g., take the wrong item or open the wrong cupboard) experiments were performed with the simulated human deviating from the plan produced by the task planner. Before the human performs an action in its plan, with a predefined probability a random (invalid) action was performed. This invalid action could be any action which is not in the person's plan and that has not already been observed; thus, if all invalid actions have already been observed, no further invalid actions can be observed.

Multiple experiments were performed for a range of probabilities; during these experiments the robot did not perform any incorrect actions. For `(packed_lunch)` and `(made_dinner)` the robot correctly opened the equipment cupboard and took a plate, and for `(made_breakfast)` opened the food cupboard, took bread and sometimes either opened the equipment cupboard and took a bowl or took a cloth. The `take(bread)` action belongs to all three goals, so no matter which action is observed its value is increased. Taking a plate is required for making salad, making a cheese sandwich and making a peanut butter sandwich; taking a bowl is required for making salad and cereal, and taking a cloth required for both making tea and making coffee. Unintentionally, the robot has performed actions which are non-distinctive, i.e., actions belonging to multiple goals' plans, thus small amounts of noisy have very little effect on the results. In future work we will investigate explicitly extracting non-distinctive actions for the robot to execute.

10. Conclusion and future work

When a robot is integrated into a smart environment, it can execute actions that help a human to achieve their goals sooner by predicting the human's next actions. Our approach applies techniques from classical planning to transform a PDDL-defined problem into an Action Graph. Action Graph node values are updated based on the observations received. Experiments show Action Graphs adequately predict actions, even when the plans of multiple interleaving subgoals are observed. Further, a simulated proof of concept demonstrated that by predicting a person's actions a robot is able to successfully assist a person by executing actions on their behalf.

We see several directions for future work. First, our approach favours the actions with shorter dependency lists due to the node value update algorithm calculating the mean value. For example, `make-tea()` has fewer dependencies than `make-coffee()`; thus, when an action common to both their dependency lists is observed, the value of `make-tea()` is increased more than the value of `make-coffee()`. Applying biases to nodes with more dependencies and weighting actions that are more unique to a plan [27] will be investigated to potentially improve the accuracy of our method. Nevertheless, for predicting the actions to reach subgoals (as well as goals), favouring actions with fewer dependencies is the desired behaviour for our system.

Second, how β and w are affected by differing action durations will be investigated. Moreover, β could be automatically adjusted based on how a human reacts to which actions the robot executes. For instance, if the robot continuously obstructs the human, β should be increased. If both the human's and robot's plans are short, increasing w will increase the chance of the robot performing an action; for long plans increasing w will have the opposite affect. Therefore, the value of w could be learnt for each domain, by measuring the length of the plans within the domain as well as if the robot acted without obstructing the human.

Performing a statistical analysis of the effect changing the Action Graph's structure has on the optimal choice for θ will be investigated. These structural changes include changing the number of dependencies each action has, the types of nodes used to connect actions to their dependencies and how unique the dependencies are. The optimal value of θ depends on which of these dependencies have been observed, i.e., if the observed dependency is unique to a correct action, θ can be set to a much lower value than if the observed dependency also belongs to a incorrect action. Moreover, such an analysis must consider the trade-off between false positive and true positive predictions.

Fourth, we intend to prevent the robot obstructing the human when it is appointed an independent goal [5,9]. To do this we will consider reasoning about the constraints between the person's and robot's plans; for example, if the person is handling an item, the robot cannot handle it at the same time. Furthermore, the person's plan may unintentionally contain actions that assist the robot, e.g., opening a door; therefore, the robot could conserve time and energy by allowing the human to perform certain actions first.

Last, our experiments assumed a rational human (i.e., followed plans produced by a task planner), and all human and robot actions took the same constant time. In future work, real-world tests will be conducted in our HomeLab[1], where we will deploy sensors, e.g., cupboard door sensors, cameras to detect items being take from in cupboards, smart electric sockets, audio sensors for recognising activities [34] etc.

Acknowledgements

Harman is an SB fellow at FWO (project number 1S40217N). Part of this research was funded via imec's RoboCure project.

Appendix A. Kitchen domain

The AND-OR trees in Figs 8, 9 and 10 represent the plans found within in the kitchen domain. To make the diagrams readable the actions to reach each hypothesis goal have been put in separate figures and `close(?container)` actions have not been included. Moreover, in the real Action Graph no action is repeated, nodes have multiple parents and only a single graph is produced. These figures provide insight on the complexity of the domain, for full details

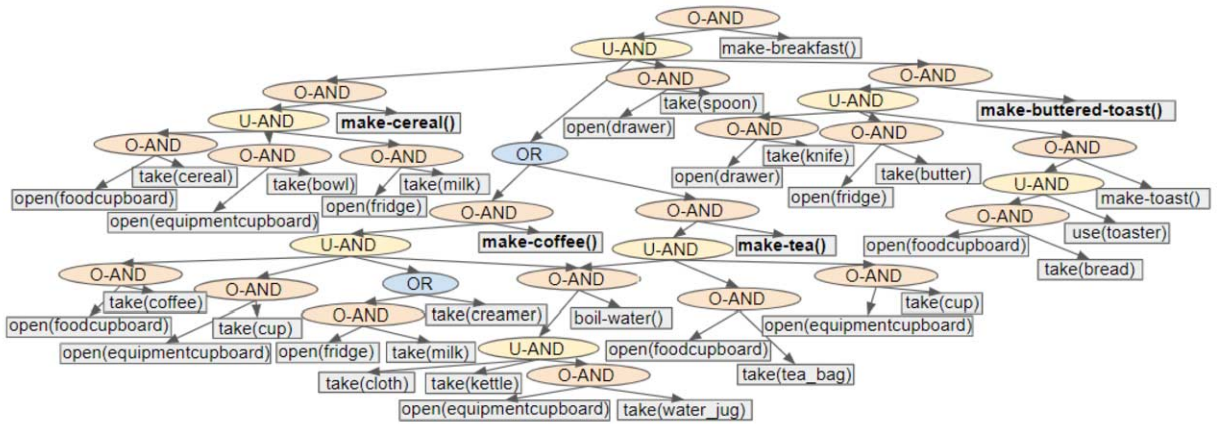


Fig. 8. The plan to make breakfast including making tea, represented as an AND-OR tree. The final actions in the subgoal's plans, used for our experiments, has been put in bold. Some action names have been shortened, e.g., `take(bread foodcupboard)` to `take(bread)`.

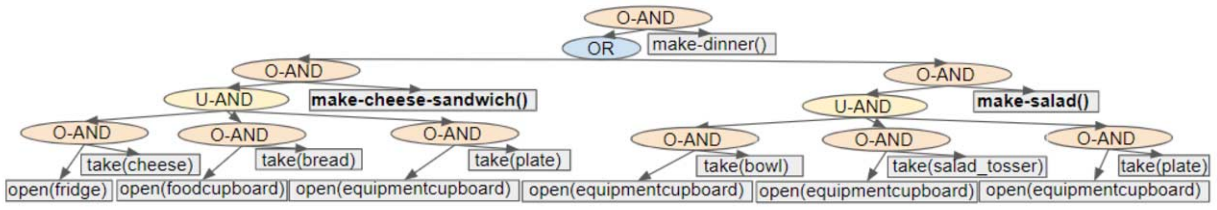


Fig. 9. The two possible plans for making dinner represented as an AND-OR tree.

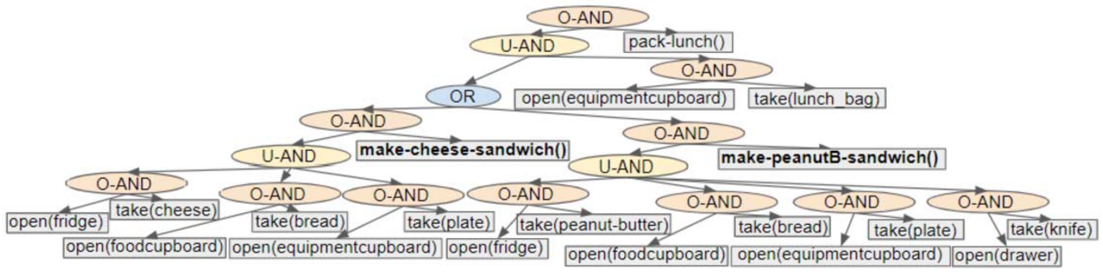


Fig. 10. The two possible plans for making lunch represented as an AND-OR tree.

the original kitchen domain produced by Ramırez et. al [30] can be download from https://sites.google.com/site/prasplanning/file-cabinet.

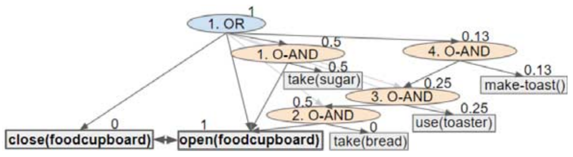
Appendix B. Reverse actions

As explained in Section 5.2, when an action is observed, its reverse actions are reset along with any action whose value was increase when the reverse action was observed. The algorithm that performs this reset procedure is provided in Algorithm 3.

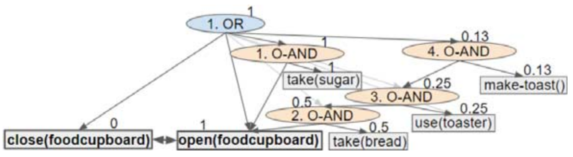
An example of how the process works is depicted in Fig. 11. After observing the action `open(food-cupboard)`, the value of its action node is set to 1 and then the reset algorithm is called on its reverse action node. As `close(foodcupboard)` has not yet been observed, no reset is required. Applying the node value update algorithms, described in Section 5.1, results in the nodes values that are shown in Fig. 11(a). The updated node values after the next observation, i.e., `take(sugar)`, was received are shown in Fig. 11(b).

Algorithm 3 Reset node value

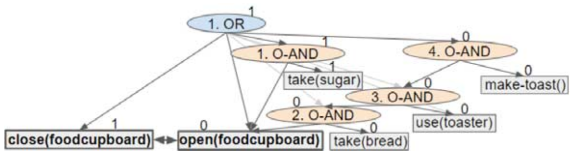
```
 1: function RESET_NODE_VALUE(node)
 2:     if node is action node and $node.value \neq 1$ then return
 3:     else if node is action node then $node.value = 0$ end if
 4:     for each parent in node.parents do
 5:         if (parent is AND node) then
 6:             RESET_CHILDREN(parent)
 7:             RESET_NODE_VALUE(parent)
 8:         else parent is OR node
 9:             parent.updateValue()                          ▷ max value of children
10:         end if
11:     end for
12: end function
13: function RESET_CHILDREN(node)
14:     for each child in node.children do
15:         if $child.value \neq 1$ and child has not been reset then
16:             if child is action node then
17:                 $child.value = 0$
18:             else if child is OR node then
19:                 child.updateValue()                       ▷ max value of children
20:             else
21:                 child.RESET_CHILDREN(child, null)
22:             end if
23:         end if
24:     end for
25:     parent.updateValue()                                  ▷ mean of children update rules
26: end function
```



(a) Node values after `open(foodcupboard)` was observed.



(b) Node values after `take(sugar)` has been observed.



(c) Node values after observing `close(foodcupboard)`.

Fig. 11. Action graphs to show the effect of reverse actions. Double ended arrows indicate two actions are the reverse of each other.

The next observation to be received, i.e., `close(foodcupboard)`, causes the `open(foodcupboard)` action node to be reset (line 3). The algorithm iterates over each of its parents, i.e., `OR` node 1, `ORDERED-AND` node 1 and `ORDERED-AND` node 2 (line 4–11). As another child of the `OR` node has a value of 1, its value is not changed. `ORDERED-AND` node 1 also remains unaltered as its subsequent child, `take(sugar)`, has already been observed. The algorithm recurs over the subsequent children of `ORDERED-AND` node 2 (lines 6 and 13–26), resulting in `use(bread)` being reset (line 17). The `ORDERED-AND` itself is then updated to the mean of its children (line 25) and its parents are iterated over (line 7 and 1–12). The process also occurs for `ORDERED-AND` nodes 3 and 4, resulting in both the `use(toaster)` and `make-toast()` action nodes being reset. After resetting the reverse action, the upwards and downwards procedures are performed. In our simple example the observed action's node (`close(foodcupboard)`) only has a single `OR` parent, i.e., the root node, whose value is already 1;

therefore, no values require updating. The final node values are shown in Fig. 11(c).

Appendix C. Experiments: Two subgoals

Our action prediction system was tested on problems in which a simulated human aimed to achieve two subgoals. The average result is provided in the main body of this manuscript, see Section 8. Table 4 shows the result for each pair of subgoals.

Appendix D. Experiments: Robot assistance

The robot assistance experiment is discussed in Section 9. In this experiment the simulated human performed the actions to reach a goal (e.g., made dinner) and the robot assisted them by executing several of the predicted actions. For when the human's goal is to make dinner and to make lunch, the action sequences (of the human and the robot) are shown Figs 12 and 13, respectively.

Table 4

Action prediction for combinations of 2 plans that achieve subgoals found in the kitchen domain. θ is set to 0.85

Goal	$\|a\|$	$\|O\|$	Planner		$p = 1$		$p = 0.75$		$p = 0.5$		$p = 0.25$		$p = 0$	
			TP	FP	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
`(made_coffee)` and `(made_buttered_toast)`	23	2	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
		7	4	0	3	0	1.4	0.0	3.6	0.4	2.6	0.8	4	2
		12	1	2	8	2	7.6	1.8	5.0	2.0	5.0	1.2	0	3
		16	1	5	4	5	4.2	5.0	4.8	4.6	4.2	4.8	0	3
`(made_coffee)` and `(made_peanut_butter_sandwich)`	22	2	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
		7	0	0	3	0	1.8	0.0	4.2	1.2	3.0	1.2	4	2
		11	7	1	7	0	6.2	1.0	6.4	1.6	2.4	2.8	0	3
		15	5	5	4	4	4.0	4.0	3.8	2.8	2.4	3.2	0	3
`(made_tea)` and `(made_buttered_toast)`	21	2	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
		6	4	0	0	0	0.0	0.0	1.2	0.0	0.6	0.0	3	0
		11	2	2	6	1	4.4	0.8	2.6	1.4	1.4	1.4	0	2
		15	2	2	4	3	3.8	2.4	2.8	2.6	3.4	3.4	3	4
`(made_tea)` and `(made_peanut_butter_sandwich)`	20	2	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
		6	0	0	0	0	0.0	0.0	1.8	0.0	1.8	0.0	2	0
		10	2	1	2	0	3.8	0.0	2.4	0.0	0.6	1.0	0	2
		14	0	2	3	1	2.6	1.4	2.6	1.4	1.6	3.2	2	6
`(made_coffee)` and `(made_cheese_sandwich)`	19	2	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
		6	0	0	0	0	0.8	0.0	1.2	0.0	3.4	1.2	5	2
		10	6	2	6	2	5.8	2.8	6.0	1.2	4.6	2.0	1	3
		13	4	7	3	6	3.6	6.6	3.4	6.0	3.0	4.8	0	3
`(made_coffee)` and `(made_salad)`	19	2	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
		6	0	0	0	1	0.8	1.0	0.6	1.0	3.2	0.6	6	0
		10	0	0	5	1	2.0	1.0	5.0	1.0	3.2	1.0	4	1
		13	4	3	4	1	4.0	1.0	3.6	1.0	3.4	1.2	1	2
`(made_salad)` and `(made_buttered_toast)`	18	2	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
		5	0	0	0	0	0.4	0.0	0.2	1.0	0.2	0.0	0	0
		9	0	0	2	2	2.6	1.4	2.8	4.0	1.6	1.0	0	0
		13	3	4	3	4	2.6	4.0	2.4	0.0	2.8	4.0	3	4
`(made_tea)` and `(made_cheese_sandwich)`	17	2	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
		5	0	0	0	0	0.0	0.0	0.8	0.0	1.2	0.0	3	0
		9	0	3	1	0	0.8	1.2	1.2	1.8	1.4	1.2	0	2
		12	3	4	3	3	2.8	2.4	2.8	3.4	2.6	2.8	0	2

Table 4

(Continued)

Goal	$\|a\|$	$\|O\|$	Planner		$p = 1$		$p = 0.75$		$p = 0.5$		$p = 0.25$		$p = 0$	
			TP	FP	TP	FP	TP	FP	TP	FP	TP	FP	TP	FP
`(made_coffee)` and `(made_tea)`	17	2	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
		5	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
		9	4	2	5	0	4.4	0.0	4.4	0.8	3.0	1.2	3	2
		12	2	2	3	0	2.6	0.0	2.8	0.8	2.4	1.2	2	2
`(made_cheese_sandwich)` and	17	2	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
`(made_buttered_toast)`		5	0	0	0	0	0.4	0.0	0.4	0.0	1.6	0.0	2	0
		9	4	1	4	1	3.4	1.0	4.0	1.0	1.8	1.2	2	2
		12	2	1	2	1	2.0	1.0	2.0	1.0	1.2	1.4	0	2
`(made_peanut_butter_sandwich)`	17	2	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
and `(made_buttered_toast)`		5	4	0	4	0	1.6	0.0	1.6	0.0	0.8	0.0	0	0
		9	4	1	4	1	4.0	1.0	4.0	0.8	3.4	0.8	1	1
		12	2	1	2	1	2.0	1.0	2.0	1.0	1.6	1.2	0	2
`(make_peanut_butter_sandwich)`	16	2	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
and `(made_salad)`		5	0	0	0	0	0.2	0.0	0.0	0.0	0.2	0.0	0	0
		8	1	1	3	1	0.6	0.2	1.0	0.0	1.2	0.0	0	0
		11	2	3	2	1	2.0	0.4	1.6	0.8	1.8	0.8	2	0
`(made_tea)` and `(made_salad)`	15	2	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
		5	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
		8	0	0	4	0	2.0	0.0	1.4	0.0	1.6	0.0	1	0
		11	3	0	2	0	2.2	0.4	2.0	0.4	1.6	0.8	0	2
`(made_cheese_sandwich)` and	15	2	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
`(made_peanut_butter_sandwich)`		5	0	0	0	0	0.8	0.0	0.8	0.0	1.2	0.0	2	0
		8	3	0	3	0	3.0	0.0	2.8	0.0	2.6	0.0	3	0
		11	1	0	1	0	1.0	0.0	1.0	0.0	1.0	0.6	1	1
`(made_cheese_sandwich)` and	13	1	0	0	0	0	0.0	0.0	0.0	0.0	0.0	0.0	0	0
`(made_salad)`		4	0	0	0	0	0.4	0.0	0.2	0.0	0.0	0.0	0	0
		7	0	0	2	0	1.8	0.4	1.8	0.4	0.4	0.8	0	0
		9	2	0	1	0	1.0	1.6	1.0	2.4	1.6	2.0	2	0

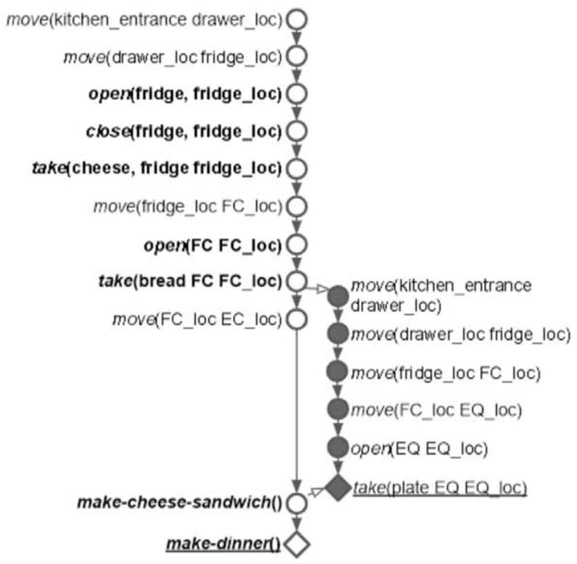


Fig. 12. The order the human performs actions when their goal is to make dinner (left) and the actions executed by the robot (right) when its initial location is `kitchen_entrance`. With the following parameters: $\theta = 0.85$, $\beta = 0.45$, $w = 0.5$.

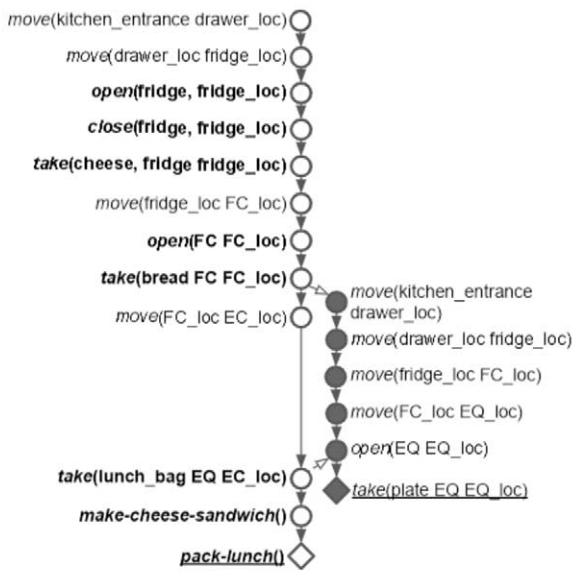


Fig. 13. The order the human performs actions when their goal is to make a pack lunch (left) and the actions executed by the robot (right) when its initial location is `kitchen_entrance`. With the following parameters: $\theta = 0.85$, $\beta = 0.45$, $w = 0.5$.

References

[1] L. Amado, R.F. Pereira, J. Aires, M. Magnaguagno, R. Granada and F. Meneguzzi, Goal recognition in latent space, in: *Proceedings of the International Joint Conference on Neural Networks, IJCNN*, IEEE, 2018, pp. 1–8, ISSN 2161-4407. doi:10.1109/IJCNN.2018.8489653.

[2] J. Baraglia, M. Cakmak, Y. Nagai, R.P. Rao and M. Asada, Efficient human-robot collaboration: When should a robot take initiative?, *The International Journal of Robotics Research* **36**(5–7) (2017), 563–579. doi:10.1177/0278364916688253.

[3] F. Bisson, H. Larochelle and F. Kabanza, Using a recursive neural network to learn an agent's decision model for plan recognition, in: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, IJCAI'15*, AAAI Press, 2015, pp. 918–924. ISBN 978-1-57735-738-4.

[4] J. Chen, Y. Chen, Y. Xu, R. Huang and Z. Chen, A planning approach to the recognition of multiple goals, *International Journal of Intelligent Systems* **28**(3) (2013), 203–216. doi:10.1002/int.21565.

[5] M. Cirillo, L. Karlsson and A. Saffiotti, Human-aware task planning: An application to mobile robots, *ACM Transactions on Intelligent Systems and Technology* **1**(2) (2010), 15:1–15:26. doi:10.1145/1869397.1869404.

[6] Y. E-Martin, M.D. R-Moreno and D.E. Smith, A fast goal recognition technique based on interaction estimates, in: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI'15*, AAAI Press, Buenos, Aires, Argentina, 2015.

[7] M. Fagan and P. Cunningham, Case-based plan recognition in computer games, in: *International Conference on Case-Based Reasoning Research and Development*, K.D. Ashley and D.G. Bridge, eds, Springer, Berlin, Heidelberg, 2003, pp. 161–170. ISBN 978-3-540-45006-1. doi:10.1007/3-540-45006-8_15.

[8] R.G. Freedman, Y.R. Fung, R. Ganchin and S. Zilberstein, Towards quicker probabilistic recognition with multiple goal heuristic search, in: *AAAI Workshops on Plan, Activity, and Intent Recognition (PAIR-18)*, 2018.

[9] R.G. Freedman and S. Zilberstein, Integration of planning with recognition for responsive interaction using classical planners, in: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI'17*, AAAI Press, 2017, pp. 4581–4588.

[10] H. Geffner and B. Bonet, *A Concise Introduction to Models and Methods for Automated Planning: Synthesis Lectures on Artificial Intelligence and Machine Learning*, 1st edn, Morgan & Claypool Publishers, 2013. ISBN 1608459691, 9781608459698. doi:10.2200/S00513ED1V01Y201306AIM022.

[11] C.W. Geib and R.P. Goldman, A probabilistic plan recognition algorithm based on plan tree grammars, *Artif. Intell.* **173**(11) (2009), 1101–1132. doi:10.1016/j.artint.2009.01.003.

[12] M. Ghallab, D. Nau and P. Traverso, *Automated Planning: Theory and Practice*, Elsevier, San Francisco, 2004.

[13] H. Harman, K. Chintamani and P. Simoens, Action Trees for scalable goal recognition in robotic applications, in: *Proceedings of the Sixth Workshop on Planning and Robotics (PlanRob)*, 2018, pp. 90–94.

[14] M. Helmert, The fast downward planning system, *J. Artif. Intell. Res.* **26** (2006), 191–246. doi:10.1613/jair.1705.

[15] S. Holtzen, Y. Zhao, T. Gao, J.B. Tenenbaum and S.-C. Zhu, Inferring human intent from video by sampling hierarchical plans, in: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2016)*, IEEE, Daejeon, South Korea, 2016, pp. 1489–1496, ISSN 2153-0866. doi:10.1109/IROS.2016.7759242.

[16] J. Hong, Goal recognition through goal graph analysis, *Journal of Artificial Intelligence Research* **15** (2001), 1–30. doi:10.1613/jair.830.

[17] P. Jonsson and C. Bäckström, State-variable planning under structural restrictions: Algorithms and complexity, *Artificial Intelligence* **100**(1) (1998), 125–176. doi:10.1016/S0004-3702(98)00003-4.

[18] H.A. Kautz, A formal theory of plan recognition, PhD thesis, University of Rochester, Department of Computer Science, 1987.

[19] H.A. Kautz and J.F. Allen, Generalized plan recognition, in: *Proceedings of the Fifth AAAI National Conference on Artificial Intelligence, AAAI'86*, Vol. 86, AAAI Press, 1986, pp. 32–37.

[20] S. Keren, R. Mirsky and C. Geib, *Plan Activity and Intent Recognition Tutorial*, AAAI, 2019, http://www.planrec.org/Tutorial/Resources_files/pair-tutorial.pdf.

[21] S.J. Levine and B.C. Williams, Concurrent plan recognition and execution for human-robot teams, in: *Proceedings of the Twenty-Fourth International Conference on International Conference on Automated Planning and Scheduling (ICAPS), ICAPS'14*, AAAI Press, 2014, pp. 490–498. ISBN 978-1-57735-660-8.

[22] S.J. Levine and B.C. Williams, Watching and acting together: Concurrent plan recognition and adaptation for human-robot teams, *Journal of Artificial Intelligence Research* **63** (2018), 281–359. doi:10.1613/jair.1.11243.

[23] W.S. Lima, E. Souto, T. Rocha, R.W. Pazzi and F. Pramudianto, User activity recognition for energy saving in smart home environment, in: *IEEE Symposium on Computers and Communication (ISCC)*, IEEE, 2015, pp. 751–757. doi:10.1109/ISCC.2015.7405604.

[24] L. Johannsmeier and S. Haddadin, A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes, *IEEE Robotics and Automation Letters* **2**(1) (2017), 41–48. doi:10.1109/LRA.2016.2535907.

[25] R. Mirsky, Y.K. Gal and S.M. Shieber, CRADLE: An online plan recognition algorithm for exploratory domains, *ACM Transactions on Intelligent Systems and Technology* **8**(3) (2017). doi:10.1145/2996200.

[26] C. Muise, S. Vernhes and P. Florian, PDDL solver in the cloud, 2018, https://bitbucket.org/planning-researchers/cloud-solver.

[27] R.F. Pereira, N. Oren and F. Meneguzzi, Landmark-based heuristics for goal recognition, in: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI'17*, AAAI Press, 2017, pp. 3622–3628.

[28] J. Rafferty, C.D. Nugent, J. Liu and L. Chen, From activity recognition to intention recognition for assisted living within smart homes, *IEEE Transactions on Human-Machine Systems* **47**(3) (2017), 368–379. doi:10.1109/THMS.2016.2641388.

[29] M. Ramírez and H. Geffner, Plan recognition as planning, in: *Proceedings of the Twenty-First International Joint Conference on Artifical Intelligence, IJCAI'09*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009, pp. 1778–1783.

[30] M. Ramírez and H. Geffner, Probabilistic plan recognition using off-the-shelf classical planners, in: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI-10*, AAAI Press, 2010, pp. 1121–1126.

[31] N. Roy, A. Misra and D. Cook, Ambient and smartphone sensor assisted ADL recognition in multi-inhabitant smart environments, *Journal of Ambient Intelligence and Humanized Computing* **7**(1) (2016), 1–19. doi:10.1007/s12652-015-0294-7.

[32] P.C. Roy, S. Giroux, B. Bouchard, A. Bouzouane, C. Phua, A. Tolstikov and J. Biswas, A possibilistic approach for activity recognition in smart homes for cognitive assistance to Alzheimer's patients, in: *Activity Recognition in Pervasive Intelligent Environments*, Atlantis Press, Paris, 2011, pp. 33–58. ISBN 978-94-91216-05-3. doi:10.2991/978-94-91216-05-3_2.

[33] G. Singla, D.J. Cook and M. Schmitter-Edgecombe, Recognizing independent and joint activities among multiple residents in smart environments, *Journal of Ambient Intelligence and Humanized Computing* **1**(1) (2010), 57–63. doi:10.1007/s12652-009-0007-1.

[34] S. Tremblay, D. Fortin-Simard, E. Blackburn-Verreault, S. Gaboury, B. Bouchard and A. Bouzouane, Exploiting environmental sounds for activity recognition in smart homes, in: *AAAI Workshop: Artificial Intelligence Applied to Assistive Technologies and Smart Environments*, 2015.

[35] S.S. Vattam, D.W. Aha and M. Floyd, Case-based plan recognition using action sequence graphs, in: *Case-Based Reasoning Research and Development*, L. Lamontagne and E. Plaza, eds, Springer International Publishing, Cham, 2014, pp. 495–510. ISBN 978-3-319-11209-1. doi:10.1007/978-3-319-11209-1_35.

[36] J. Wu, A. Osuntogun, T. Choudhury, M. Philipose and J.M. Rehg, A scalable approach to activity recognition based on object use, in: *Proceedings of the Eleventh IEEE International Conference on Computer Vision*, IEEE, 2007, pp. 1–8, ISSN 1550-5499. doi:10.1109/ICCV.2007.4408865.

[37] K. Yordanova, F. Krüger and T. Kirste, Context aware approach for activity recognition based on precondition-effect rules, in: *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, Vol. 00, IEEE, 2012, pp. 602–607. doi:10.1109/PerComW.2012.6197586.

[38] K. Yordanova, S. Lüdtke, S. Whitehouse, F. Krüger, A. Paiement, M. Mirmehdi, I. Craddock and T. Kirste, Analysing cooking behaviour in home settings: Towards health monitoring, *Sensors* **19**(3) (2019). doi:10.3390/s19030646.

[39] Y. Zhang, V. Narayanan, T. Chakraborti and S. Kambhampati, A human factors analysis of proactive support in human-robot teaming, in: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2015, pp. 3586–3593. doi:10.1109/IROS.2015.7353878.