Action trees for scalable goal recognition in robotic applications

Helen Harman, Keshav Chintamani, Pieter Simoens
Department of Information Technology - IDLab,
Ghent University - imec,
Technologiepark 15, B-9052 Ghent, Belgium
{firstname.surname}@ugent.be

Abstract

Robots are being deployed in a wide range of environments to assist humans with their daily activities. To assist a person, and avoid obstructing them when executing a different task, a robot needs to know what the intentions of the people are. In this short paper we present an early version of our work, in which we focus on goal recognition using techniques from classical symbolic planning to form an *Action Tree*. We present results which show improved goal recognition times, without compromising on accuracy.

Introduction

Increasingly robots are being developed to work alongside and help humans, therefore it is essential for a robot to understand the intentions of the humans. In many situations there are multiple ways the same aim can be achieved. Thus, in order to assist the person a robot will need to recognise both the goal and the intended plan. However, humans are likely to switch between goals and leave goals partially completed causing real world intention recognition to be a challenging problem. In addition to this, noisy erroneous sensor observations may cause further problems. Our long-term aim is to enable robots to provide assistant to humans with their daily activities, by recognising a person's goal and how they intend to reach that goal (i.e. their plan).

In this paper we focus on single-goal recognition, as a first step. This is to show our algorithm's potential advantages for use in intention recognition in robotic applications. We propose transforming the planning problem into an *Action Tree* (i.e. AND-OR tree with some temporal constraints). This allows the dependencies between different actions to be represented, and the most likely plans as well as goals can be extracted.

A well-studied approach to intention recognition is searching through a dictionary/library of predefined plans (Zhuo and Li 2011). (Holtzen et al. 2016) take a similar approach to us, as they use a Temporal AND-OR tree. However, their probability update rules differ from ours, and we do not use a dictionary, as we aim to allow more flexibility in the way a person's intentions are modelled (Ramırez and Geffner 2010).

One approach is through training a model on humans' intentions using a set of training data, such as HMM (Singla, Cook, and Schmitter-Edgecombe 2010) and RNN (Bisson, Larochelle, and Kabanza 2015). These types of approaches can require a lot of time being spent on manually labelling data and can produce models which only work on data similar to the training set (Yordanova, Krüger, and Kirste 2012).

Due to the above disadvantages we have opted to use a classical planning approach to intention recognition, which in some literature is referred to as goal/plan recognition as planning (Sohrabi, Riabov, and Udrea 2016) or inverse planning (Ramırez and Geffner 2010). In (Ramırez and Geffner 2010), (Chen et al. 2013) and (Freedman and Zilberstein 2017) a planner must be called twice for every possible goal, which is unscalable to large state-spaces.

(Pereira, Oren, and Meneguzzi 2017) significantly reduce the recognition time through the use of landmarks, i.e. actions that must always be performed for a goal to be reached. We will compare our approach to this and show we have improved the scalability of goal recognition. In (Freedman et al. 2018) an algorithm to speed-up recognition time, by only using a single call to the planner, has been proposed but not yet implemented, therefore in future work we would look at also comparing to this approach.

We begin by presenting a brief description of the planning algorithm we have adapted. Then, we describe our approach to goal recognition. Finally, we give our preliminary result.

Background

Traditionally a planning problem is formally defined as $P = (F, I, A, G)$. Where F is a set of atoms, $I \subset F$ is the initial state, $G \subset F$ is a goal state, and A is a set of actions along with their preconditions and effects (Ramırez and Geffner 2010). A task planner is used to find the least costly sequence of actions required to reach the goal state. Often these planning problems are written in Planning Domain Definition Language (PDDL).

In Fast Downward (FD) (Helmert 2006) the planning problem is first translated into SAS+ (a "multivalued planning tasks" representation). Actions and states which are impossible to reach from the goal are removed during this translation. From this the causal relationships between state variables (i.e. causal graph) and how the variables change state i.e. Domain Transition Graphs (DTGs) are determined. Every variable has its own DTG. The causal graph and DTG are used during the search for a plan.

Goal recognition is often viewed as the inverse of planning i.e. $T = (F, I, A, O, \mathcal{G})$ where $\mathcal{G}$ is the set of all possible goals and O is the sequence of observed actions (Ramırez and Geffner 2010). In this paper, we aim to find the probability of each $G \in \mathcal{G}$.

Dataset

For evaluating our approach we use the datasets produced by (Pereira, Oren, and Meneguzzi 2017), which are based on the problems used for the International Planning Competition (IPC). Each dataset contains a PDDL domain and template (i.e. problem without a goal) file, a list of possible goals $\mathcal{G}$, and a sequence of observations $O \subset A$. To check the results produced by goal recognition the real goal is provided.

In Figure 2 we show the Action Tree for the *Kitchen* dataset. The next section describes how it has been created and how the probability of an action being performed is updated.

Method

Our software starts by creating a PDDL problem file, consisting of the template plus a goal state containing all of the possible goals $\mathcal{G}$ in an `or` statement. The PDDL domain and problem file is then transformed into a set of DTGs. Once the Action Tree has been created from the DTGs, the probability of each action $a \in A$ appearing in the person's plan is updated based on the observations O. The tree is then searched for the most likely actions which result in each of the possible goals being reached. The goal(s) with the highest probability are returned, i.e. the *candidate goals* $\mathcal{C} \subset \mathcal{G}$. Multiple candidate goals could be returned as several goals can be equally likely. An overview of our system is shown in Figure 1. We use the term *dependencies* to mean the actions that must be performed before another action can be performed.

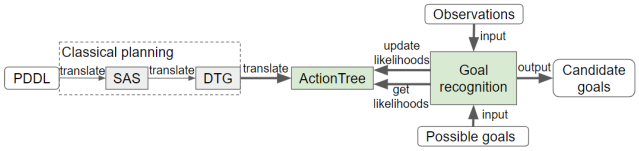

Figure 1: The most important classes in our system are depicted in green boxes; grey boxes show the steps performed by the original task planner FD (Helmert 2006).

To describe our method we use the *Kitchen* dataset. In this dataset there are 3 possible goals: `made_breakfast`, `lunch_packed` and `made_dinner`. For each of these goals there are multiple plans which can be used to reach that goal, e.g. for `lunch_packed` a person must always perform the `take(lunch_bag)` action and has the option of either perform the `activity-make-peanut-butter-sandwich` or `activity-make-cheese-sandwich` action.

Action tree creation

To perform goal recognition, we transform the DTGs into an Action Tree, in which leaf nodes are actions and all other nodes are: `OR` nodes in which one or more of the sub-trees must be performed; `UNORDERED-AND` nodes where all sub-trees are performed in any order, and `ORDERED-AND` nodes for which all sub-trees must be performed in order.. All examples used in this section are shown in the Action Tree depicted in Figure 2. Action (leaf) nodes and `ORDERED_AND` nodes can have multiple parent nodes, as the Action Tree only contains one action node per action. Unless otherwise stated, we always use the term parent(s) to refer to the direct parent(s) of a node. A tree is initialised with an `OR` node as the root, this root remains the same and will receive a new child for every action inserted into the tree.

Each DTG describes how a variable changes state. Multiple labels are given to transitions with multiple possible preconditions, e.g. the preconditions for `activity-Pack-lunch` require either `(made_cheese_sandwich)` or `(made_peanut_butter_sandwich)` to be true. `activity-Pack-lunch` has the effect `lunch_packed`, therefore the transition to `lunch_packed` being true will have at least two labels.

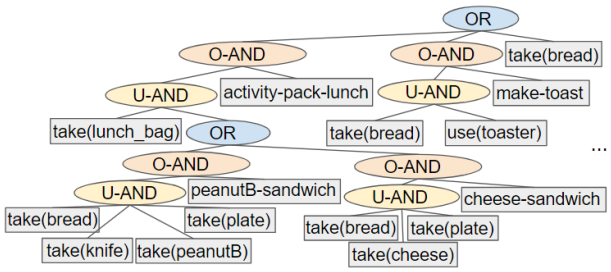

Figure 2: Small section of the action tree created from the kitchen domain. O-AND stands for `ORDERED-AND` and U-AND is `UNORDERED-AND`. For readability some action names have been shortened, e.g. activity-make-peanut-butter-sandwich has been shortened to peanutB-sandwich. Note, in this figure nodes have been repeated to represent that they have multiple parent nodes.

Our system iterates through all the DTG transitions for all of the variables and adds each transition (i.e. action) to the tree. Actions which do not have any preconditions are appended to the root node's children, e.g. `take(bread)`. Actions with dependencies/preconditions are added after all of their dependencies are, e.g. `take(lunch_bag)`, `activity-make-cheese-sandwich` and `activity-make-peanut-butter-sandwich` are added to the tree before `activity-pack-lunch` is.

When an action has dependencies, an `UNORDERED-AND` node is created containing all the dependencies as its children. These children will now have at least two parents, the `UNORDERED-AND` and the root node. The `UNORDERED-AND` node along with the action itself are added to a new `ORDERED-AND` node, which is appended to the root node's children. If the dependencies have dependencies, then a direct child of the `UNORDERED-AND` node will be an `ORDERED-AND` node.

For example, the action `activity-make-cheese-sandwich` requires the actions `take(bread)`, `take(cheese)` and `take(plate)` to be performed first, however it does not matter what order the required actions are performed, therefore they become the chil-

dren of an UNORDERED_AND node; which along with the activity-make-cheese-sandwich action is set as the ORDERED_AND node's children. Note, if a node has an ORDERED-AND node as its parent it can only have one parent.

When a PDDL action contains a precondition which has an or statement or multiple actions exist which result in the same state being reached, the DTG transition will have multiple labels. This results in OR nodes being inserted into the tree. For example, to complete the preconditions of the action activity-pack-lunch the take(lunch_bag) and either activity-make-peanut-butter-sandwich or activity-make-cheese-sandwich must have been performed.

Updating probabilities based on observations

All action nodes are initialised with a probability of 0.5, as they are all equally likely to appear or not appear in a person's plan. We experimented with different initial values but found this made little difference to our results. In future work we intend to experiment with multiple interleaving goals, which this value may have a greater impact on.

When an observation $o \in O \subset A$ is received that action's probability is set to 1, this is shown in line 3 of Algorithm 1. The action node's parents are then updated (lines 10-12). If a parent is an OR node its probability is set to the maximum probability of its children (line 5), otherwise it is set to the mean probability of its children (line 7). This algorithm recurses (line 11) until the root node is reached (line 13). It does not matter in which order a node's parents are updated.

Algorithm 1 Update node probability upwards

```
1:  function UPDATE_PROBABILITY_UPWARDS(node)
2:      if node is an action node then          ▷ The observed action
3:          node.probability = 1.0.
4:      else if node is an OR node then
5:          node.probability = max(children).
6:      else node is an AND node
7:          node.probability = mean(children)
8:      end if
9:      for each parent in node.parents do
10:         UPDATE_PROBABILITY_UPWARDS(parent)
11:     end for
12:     if node is an action node then          ▷ The observed action
13:         UPDATE_PROBABILITY_DOWNWARDS(root)
14:     end if
15: end function
```

We considered using product, rather than mean but found the size of the sub-trees had a much larger effect on the probability of a goal (i.e. strongly favours shorter plans), therefore we opted to use mean as this achieved better results. The maximum probability is used for OR nodes as it does not matter which one of its children have been (partially) executed.

To set the probability of an action appearing in the subsequently performed actions we then traverse down the tree (depth-first) using Algorithm 2. If the current node is an AND node (line 2) and its child's probability is lower, then the child's probability is assigned the AND node's probability (lines 3-5). The direct children of OR nodes are not updated.

Algorithm 2 Update node probability downwards

```
1:  function UPDATE_PROBABILITY_DOWNWARDS(node)
2:      if node is an AND node then
3:          for each child in node.children do
4:              child.probability = max(child, this)
5:          end for
6:      end if
7:      for each child in node.children do
8:          UPDATE_PROBABILITY_DOWNWARDS(child)
9:      end for
10: end function
```

Goal recognition

Each goal $G_i \in \mathcal{G}$ contains one or more atoms $G_i \subset F$. For each atom $f \in G_i$ we find the most likely action whose effects contain f; and find the average over all atoms in G_i. This is shown in Equation 1.

$$p(G_i) = \frac{\sum_{f \in G_i} \max(p(a_{1f \in eff}), ..., p(a_{Nf \in eff}))}{|f \in G_i|} \quad (1)$$

Where $p(G_i)$ is the probability of the i-th goal in $\mathcal{G}$ and $p(a_{1f \in eff})$ is the probability of an action $a_1 \in A$ whose effects contain f. If $p(G_i) \equiv \max(p(G_1), p(G_2), ..., p(G_N))$ then G_i is added to the set of candidate goals $\mathcal{C}$.

Preliminary results

We ran our goal recognition, and the goal completion heuristic from (Pereira, Oren, and Meneguzzi 2017), on a dataset they produced. For both approaches we only consider the most likely goals as being in the set of candidate goals (i.e. the threshold value described by Pereira et al. is set to 0).

The dataset consists of 15 domains and a total of 6313 goal recognition problems; which include problems where 10%, 30%, 50%, 70% and 100% of observations are provided. The goal recognition times for each domain are given in Table 1 and the accuracy is presented in Table 2. $|\mathcal{C}|$ is the number of candidate goals, and the accuracy A is determined by the number of times the correct goal appears in the list of candidate goals.

On all plan recognition problems our approach is faster than that of (Pereira, Oren, and Meneguzzi 2017). Overall our approach took 1727s to run on all plan recognition problems, whereas their approach took 7798s.

Table 1: Recognition times per domain. All times are in seconds. As planning problems can greatly vary in size we show the standard deviation.

		Ours		Pereira et al.	
Domain	$\|probs\|$	$\sum t$	$\overline{t} \pm$ std	$\sum t$	$\overline{t} \pm$ std
miconic	364	125.44	0.34 ± 0.26	546.90	1.50 ± 1.07
sokoban	364	140.09	0.38 ± 0.14	579.63	1.59 ± 0.49
satellite	364	127.15	0.35 ± 0.23	621.45	1.70 ± 1.19
logistics	673	170.24	0.25 ± 0.25	1089.15	1.61 ± 1.05
ferry	364	60.86	0.17 ± 0.07	258.40	0.71 ± 0.17
rovers	364	186.93	0.51 ± 0.33	545.47	1.49 ± 0.78
intrusion-detection	465	54.97	0.12 ± 0.01	331.35	0.71 ± 0.07
kitchen	75	8.10	0.11 ± 0.00	41.64	0.55 ± 0.07
easy-ipc-grid	673	127.58	0.19 ± 0.05	743.55	1.10 ± 0.37
blocks-world	1076	205.36	0.19 ± 0.07	941.76	0.88 ± 0.57
depots	364	134.22	0.37 ± 0.17	481.02	1.32 ± 0.30
zeno-travel	364	166.02	0.46 ± 0.16	615.76	1.69 ± 0.65
dwr	364	110.78	0.30 ± 0.06	517.21	1.42 ± 0.37
campus	75	8.74	0.12 ± 0.00	45.42	0.61 ± 0.06
driverlog	364	100.80	0.28 ± 0.17	438.87	1.21 ± 0.71
ALL	6313	1727.27	0.27 ± 0.20	7797.56	1.24 ± 0.84

On average when 10% of observations are provided our approach has more candidate goals and therefore a higher accuracy. As the number of observations increases the number of candidate goals decreases. For (Pereira, Oren, and Meneguzzi 2017) the number of candidate goals does not decrease by much, however the accuracy increases as the number of observations increases. There are some domains which are exceptions to this trend, such as the kitchen domain where our approach produces fewer candidate goals.

Conclusion and future work

In this paper we presented an early version of our intention recognition system, where we focus on single-goal recognition. DTGs are translated into an Action Tree which is used to predict the probability of a person performing an action. We compared our approach to (Pereira, Oren, and Meneguzzi 2017) and found our approach is quicker and performs equally well in terms of accuracy.

We intend to extract the most likely actions a human will perform from the tree. This will enable a robot to assist the person (e.g. open doors, fetch objects, provide instructions) and avoid obscuring the person when performing a different task within the same environment. In the case of assisting a person, we will investigate how confident the recognition should be before the robot attempts to give assistance.

In future work we will also provide experimentation results for datasets containing invalid and missing observations caused by noisy sensor readings. Additionally, we will test our approach on multiple interleaving and concurrent goals. In dynamically changing environments multiple humans act continuously, including leaving and returning to the environment. Therefore, rather than ending when a goal (or set of goals) is reached, we will investigate how the intentions of a human can be continuously updated. To do this we will experiment with decaying the probability of actions the human has performed, when they are no-longer part of the person's intended plan.

Acknowledgements

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

Table 2: Accuracy for the different domains when different percentages of observations are known. To save space we do not show results for 100% of observations.

			Ours		Pereira et al.	
Domain	$\overline{\|\mathcal{G}\|}$	**Obs %**	$\|\mathcal{C}\|$	A	$\|\mathcal{C}\|$	A
miconic	6	10	4.40	0.96	1.46	0.69
		30	2.64	0.96	1.15	0.98
		50	1.88	0.96	1.02	0.99
		70	1.23	0.96	1.01	1.00
sokoban	7.14	10	5.35	0.93	2.10	0.55
		30	2.73	0.82	1.40	0.58
		50	2.26	0.86	1.35	0.71
		70	1.45	0.95	1.08	0.86
satellite	6.43	10	2.65	0.87	2.18	0.70
		30	1.51	0.87	1.45	0.86
		50	1.17	0.89	1.29	0.94
		70	1.11	0.96	1.05	0.99
logistics	10.46	10	6.74	0.96	2.01	0.63
		30	3.39	0.98	1.34	0.86
		50	1.91	0.97	1.21	0.95
		70	1.24	0.99	1.10	0.97
ferry	7.57	10	3.51	0.98	1.45	0.64
		30	1.50	0.88	1.15	0.86
		50	1.26	0.92	1.07	0.94
		70	1.12	1.00	1.00	0.96
rovers	6	10	2.73	0.87	1.82	0.67
		30	1.31	0.85	1.36	0.82
		50	1.12	0.96	1.12	0.89
		70	1.00	0.98	1.05	1.00
intrusion-detection	16.67	10	1.39	0.73	1.37	0.74
		30	1.05	0.96	1.03	0.95
		50	1.01	0.99	1.03	1.00
		70	1.00	0.99	1.00	1.00
kitchen	3	10	1.00	0.80	3.00	1.00
		30	1.00	0.93	2.60	1.00
		50	1.00	0.93	2.60	1.00
		70	1.00	0.93	2.33	1.00
easy-ipc-grid	8.66	10	7.67	1.00	2.58	0.67
		30	6.03	1.00	1.65	0.82
		50	3.97	1.00	1.18	0.91
		70	3.12	1.00	1.07	0.97
blocks-world	20.28	10	7.07	0.62	1.26	0.44
		30	1.65	0.62	1.17	0.56
		50	1.20	0.74	1.13	0.63
		70	1.19	0.91	1.15	0.84
depots	8.86	10	4.42	0.90	1.31	0.39
		30	2.44	0.93	1.15	0.67
		50	1.69	0.98	1.11	0.85
		70	1.44	0.99	1.01	0.94
zeno-travel	6.86	10	3.33	0.92	1.43	0.45
		30	2.11	0.90	1.40	0.79
		50	1.27	0.96	1.15	0.82
		70	1.05	1.00	1.10	0.98
dwr	7.29	10	2.55	0.61	1.20	0.38
		30	1.51	0.75	1.10	0.64
		50	1.38	0.85	1.06	0.73
		70	1.15	0.89	1.05	0.90
campus	2	10	1.93	0.93	1.13	0.87
		30	1.93	0.93	1.13	0.87
		50	1.80	0.87	1.13	0.93
		70	1.73	0.93	1.00	1.00
driverlog	7.14	10	3.17	0.83	1.29	0.45
		30	1.67	0.73	1.24	0.60
		50	1.40	0.90	1.29	0.77
		70	1.19	0.95	1.24	0.93
ALL	10.43	10	3.86	0.86	1.71	0.62
		30	2.16	0.87	1.36	0.79
		50	1.62	0.92	1.25	0.87
		70	1.33	0.96	1.15	0.96

References

Bisson, F.; Larochelle, H.; and Kabanza, F. 2015. Using a recursive neural network to learn an agent's decision model for plan recognition. In *IJCAI*, 918–924.

Chen, J.; Chen, Y.; Xu, Y.; Huang, R.; and Chen, Z. 2013. A planning approach to the recognition of multiple goals. *Intl Journal of Intelligent Systems* 28(3):203–216.

Freedman, R. G., and Zilberstein, S. 2017. Integration of planning with recognition for responsive interaction using classical planners. In *AAAI*, 4581–4588.

Freedman, R. G.; Fung, Y. R.; Ganchin, R.; and Zilberstein, S. 2018. Towards quicker probabilistic recognition with multiple goal heuristic search.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

Holtzen, S.; Zhao, Y.; Gao, T.; Tenenbaum, J. B.; and Zhu, S.-C. 2016. Inferring human intent from video by sampling hierarchical plans. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ Intl Conf on*, 1489–1496. IEEE.

Pereira, R. F.; Oren, N.; and Meneguzzi, F. 2017. Landmark-based heuristics for goal recognition. In *Thirty-First AAAI Conf on Artificial Intelligence (AAAI-17)*. AAAI Press.

Ramırez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *Proceedings of the Conf of the Association for the Advancement of Artificial Intelligence (AAAI 2010)*, 1121–1126.

Singla, G.; Cook, D. J.; and Schmitter-Edgecombe, M. 2010. Recognizing independent and joint activities among multiple residents in smart environments. *Journal of ambient intelligence and humanized computing* 1(1):57–63.

Sohrabi, S.; Riabov, A. V.; and Udrea, O. 2016. Plan recognition as planning revisited. In *IJCAI*, 3258–3264.

Yordanova, K.; Krüger, F.; and Kirste, T. 2012. Context aware approach for activity recognition based on precondition-effect rules. In *PERCOM Workshops, 2012 IEEE Intl Conf on*, 602–607. IEEE.

Zhuo, H. H., and Li, L. 2011. Multi-agent plan recognition with partial team traces and plan libraries. In *IJCAI*, volume 22, 484.