Action Graphs for Goal Recognition Problems with Inaccurate Initial States (Student Abstract)

Helen Harman*, **Pieter Simoens**
Department of Information Technology – IDLab, Ghent University – imec,
Technologiepark 126, B-9052 Ghent, Belgium
{helen.harman, pieter.simoens}@ugent.be

Abstract

Goal recognisers attempt to infer an agent's intentions from a sequence of observations. Approaches that adapt classical planning techniques to goal recognition have previously been proposed but, generally, they assume the initial world state is accurately defined. In this paper, a state is inaccurate if any fluent's value is unknown or incorrect. To cope with this, a cyclic Action Graph, which models the order constraints between actions, is traversed to label each node with their distance from each hypothesis goal. These distances are used to calculate the posterior goal probabilities. Our experimental results, for 15 different domains, demonstrate that our approach is unaffected by an inaccurately defined initial state.

Introduction

Goal recognisers are an important component of intelligent systems that aim to assist or thwart actors; however, there are many challenges to overcome. For instance, the defined initial state of the environment could be inaccurate. In this paper, a state is inaccurate if any fluent's (i.e, non-static variable's) value is unknown or incorrect; e.g., if an item is occluded, its location is indeterminable, thus possibly defined incorrectly. To our knowledge, this remains an open challenge for goal recognisers that stem from planning methods.

Knowledge driven (symbolic) goal recognition (GR) approaches can be divided into those that parse a library of plans (Geib and Goldman 2009) and those that take classical planning languages, such as Planning Domain Definition Language (PDDL), as input (Ramírez and Geffner 2010). Our approach generates a graph structure, similar to those used by some recognition as parsing methods, from a PDDL defined GR problem.

Action Graphs model the dependencies, i.e., order constraints, between all actions. After generating an Action Graph, its nodes are labelled with their distance from each hypothesis goal. These two processes are performed offline. For each observation, the online process updates the goal probabilities based on either the observed action's distances from the goals or the change in distance.

*Harman is an SB fellow at FWO (1S40217N).

Background

To create a concise, grounded representation of a problem, a PDDL defined problem is often converted into a multi-valued representation (Helmert 2006). Once converted, a GR problem can be defined as $T = (F, I, A, O, \mathcal{G})$, where F is a set of fluents, I the initial state, A a set of actions and $\mathcal{G}$ is the set of possible (hypothesis) goals (Ramírez and Geffner 2010). O is a sequence of discrete observations (i.e., actions) to reach a goal $G \in \mathcal{G}$. An action $a \in A$ has effects $a_{eff} \subseteq F$ and preconditions $a_{pre} \subseteq F$. GR approaches return a set of candidate goals $\mathcal{C}$, containing the goals with the maximum probability/value.

In our system, the set of actions A contains all groundings of the action definitions whose static preconditions are within the PDDL defined initial state, including those that are unreachable given the value of the fluents in I. For instance, when the observee's location is missing from I (e.g., because it is unknown) all groundings of the move(?1 ?2) action definition are still in A, as long as the locations, i.e, groundings of ?1 and ?2, are defined as being adjacent.

Method

Action Graphs are constructed of action nodes and operator nodes, which includes DEP (short for dependencies), UNORDERED-AND, OR and ORDERED-AND nodes. An action's dependencies are conveyed through its connections (via operator nodes) to other actions. A dependency is an action that sets one or more of a dependant's preconditions. For instance, $a2$ is a dependency of $a1$ if $a2$ fulfils one (or more) of $a1$'s preconditions, i.e., $a2_{eff} \cap a1_{pre} \neq \emptyset$. Thus, an action's set of dependencies is formally defined as: $D(a \in \mathcal{A}) = \{a' \mid (a'_{eff} \cap a_{pre}) \neq \emptyset\}$. The different node types, depicted in Figure 1, are described below.

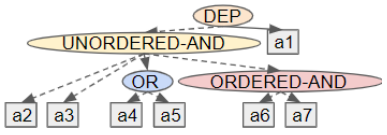

Figure 1: The different order constrains on actions that achieve $a1$'s preconditions, i.e., $(a2 \wedge a3 \wedge (a4 \vee a5) \wedge (a6 \prec a7)) \prec a1$. Solid arrows point to the dependant and dashed arrows point to the dependencies.

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

An Action Graph is initialised with an OR node as the root; then each action ($a \in A$) is inserted into the graph in turn by connecting it to its dependencies. Finally, the graph is adjusted so only the Goal Actions' parent DEP nodes are connect to the root. Goal Actions are actions whose effect(s) reach a goal; if multiple actions are required to reach a goal, an auxiliary Goal Action is inserted.

Each node has a set of distances associated with it, which indicate how far the node is from each goal, i.e., the number of DEP and ORDERED-AND nodes that must be traversed to get from the Goal Action's parent to the node in question. These distances are set by performing a breadth-first traversal (BFT) from each Goal Action. The same node could be visited multiple times during a BFT; however, if the current distance/count is greater than (or equal to) the node's already assigned distance, it is not reprocessed.

As an action could appear in a plan multiple times, some nodes require multiple distances for the same goal; this is the case for the descendants of ORDERED-AND nodes' right-branch. Therefore, a node contains a map for each goal, from the last traversed ORDERED-AND to the node's distance from the goal via the ORDERED-AND node. When the right-branch of the ORDERED-AND node has been fully observed, the distance of the node, returned when calling a get distance method, will be the distance associated with that ORDERED-AND node.

The hypothesis goals have a uniform prior probability, which is updated when an observation is received. If the previous o^{t-1} and the current (just received) o^t observations are linked via a DEP or ORDERED-AND node, each goal's probability is updated based on its change in distance, i.e., $P(G) = norm(P(G)(1+\sigma(dis(o^{t-1},G) - dis(o^t,G))))$. Otherwise, the probability of the goals closest to the observation are increased and the others decreased, i.e., $P(G) = norm(P(G)(1+(\frac{dis(o^t,G)}{\sum_{G' \in \mathcal{G}} dis(o^t,G')})))$. If either observation is not an (indirect) dependency of the goal, the goal's probability is (likely) decreased, i.e., $P(G) = norm(P(G)(1+0))$. So that our system can recover from noisy (incorrect) observations, a goal's probability is never set to 0.

Results and Discussion

Our approach and the goal completion heuristic (h_{gc}) of Pereira, Oren, and Meneguzzi (2017) was ran on a dataset, containing the first 10 %, 30 %, 50 %, 70 % and 100 % of observations, encompassing 15 difference domains. Differing percentages of fluents where set to a randomly selected, incorrect initial value. The resulting F1-Scores, shown in Figure 2, demonstrate that our approach is unaffected by an inaccurate initial state; whereas the accuracy of h_{gc} declines.

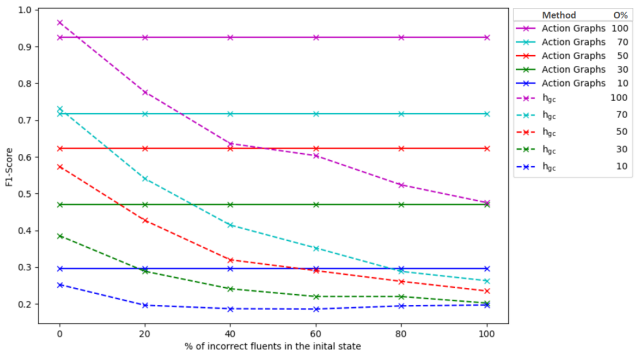

Figure 2: Graph showing the effect increasing the amount of incorrect fluents in the initial state had on the accuracy of our Action Graph approach (solid lines) and h_{gc} by Pereira, Oren, and Meneguzzi (2017) (dashed lines). Each colour indicates a different % of observations.

Conclusion

An Action Graph represents the dependencies, i.e., order constraints, between actions. Each node is labelled with the number of DEP and ORDERED-AND nodes, traversed to reach it from each Goal Action. This distance is used to update the goals' probability when an observation is received. Experimental results demonstrate that our approach is unaffected by inaccuracies within the defined initial state.

References

Geib, C. W., and Goldman, R. P. 2009. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence* 173(11):1101 – 1132.

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

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

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