Solving Navigation-Based Goal Recognition Design Problems with Action Graphs

Helen Harman and **Pieter Simoens**
Department of Information Technology - IDLab
Ghent University - imec
Technologiepark 15, B-9052 Ghent, Belgium
{helen.harman, pieter.simoens}@ugent.be

Abstract

To proactively assist humans robots require the ability to recognise a human's goal. Unfortunately, as plans can often start with the same actions, i.e. are initially non-distinctive, it is often impossible to determine which goal a human is aiming to achieve until they have neared their goal. By modifying the environment we aim to force humans into revealing their goal sooner without increasing the cost of the optimal plan to any of the possible goals. Previously proposed approaches, that apply classical planning techniques to this problem, are very computationally expensive. This paper presents an early version of our work, in which we propose transforming goal recognition design problems into Action Graphs. Experiments show that our approach is able to force humans into revealing their goal sooner in environments with various numbers of goals and of differing sizes in an average time of 1.42 seconds, whereas a current state-of-the-art approach often breaches a 10 minute timeout.

1 Introduction

Increasingly robots and smart devices are being deployed to help humans with the daily activities. To proactively provide assistance it is essential for a robot to understand the intentions of humans. Besides robotic environments, discovering the intentions of humans can be beneficial in many different situations including: monitoring where people are walking in an airport for security reasons (Keren, Gal, and Karpas 2014); crowd monitoring at events such as festivals; and in hospitals to determine where a patient or staff member is heading.

Often a person's intentions cannot be determined until they near their goal, this is due to multiple goals being reachable by initially performing identical actions. By modifying the initial environment the number of actions a human performs before their intentions are revealed can be reduced. We have identified 2 ways in which an environment, prior to humans occupying it, can be modified: 1) prevent actions from being performed, e.g. by blocking a person from navigating between two locations, and 2) modify the state of an object, e.g. change its location. In this paper we focus on the first aspect, which in realistic human occupied environments is, as far as we are aware, only applicable to navigation domains.

To clarify the problem we provide an example. Figure 1 shows two goals, these goals could indicate the location of e.g. a coffee machine and fridge in a smart house, or a shop and terminal in an airport. At worst the plans for these two goals have a non-distinctive plan prefix containing 3 actions, i.e. the Worst Case Distinctiveness (WCD) for this environment is 3. By placing an obstacle (e.g. a wall) at a strategically chosen position, the action to move from position (2,3) to position (2,2) becomes impossible and the WCD of the sample environment is reduced to 0. The term goal recognition design and the WCD metric were introduced by Keren, Gal, and Karpas (2014).

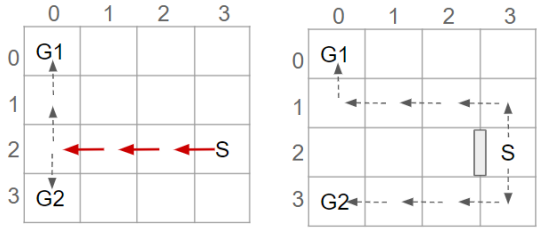


Figure 1: In this grid-based navigation example a human can only move horizontally and vertically. There are multiple optimal plans to each of the goals, but for readability only a single plan to each goal is shown (indicated with arrows). The longest non-distinctive plan prefix from the start location (S), which is part of an optimal plan for the two goals (G1 and G2) is indicated by red arrows. The WCD for the environment on the left is 3. The image on the right shows that by preventing the human from perform a single action the goal of the human can be determined from their first action, i.e. the WCD is 0.

Goal recognition design is a complex problem. Rather than finding a single optimal solution to reach a goal like in task planning, *all* possible optimal plans that have non-distinctive prefixes are required. Thus, current approaches to goal recognition design (Wayllace et al. 2016; Son et al. 2016; Wayllace et al. 2018; Keren, Gal, and Karpas 2018) are computationally expensive. In this paper we present a novel approach, which transforms goal recognition design

problems into Action Graphs. Non-distinctive plan prefixes are extracted from the graph and processed to determine the actions whose removal reduces the WCD without increasing the cost of the optimal plan from the initial state to any of the goal states. The removed actions indicate the locations obstacles should be placed within the environment to make goal recognition less challenging.

Harman, Chintamani, and Simoens (2018) introduced the concept of creating Action Trees using techniques from classical planning, to perform goal recognition. We build on the work described in their paper. As the model produced did not work well for navigation domains, we have changed how the model (i.e. Action Tree/Graph) is generated.

Section 2 provides an overview of related work. How action Graphs are generated, the non-distinctive plan prefixes found and the WCD reduced is described in Section 4. Our initial experiments that compare our approach to a state-of-the-art approach (Keren, Gal, and Karpas 2014), are presented and discussed in Section 5.

2 Related Work

There are several different methods for intention recognition, including searching a dictionary/library of predefined plans (Goldman, Geib, and Miller 1999; Zhuo and Li 2011; Holtzen et al. 2016); processing labelled data (e.g. videos) to train a model (Singla, Cook, and Schmitter-Edgecombe 2010; Bisson, Larochelle, and Kabanza 2015); and solving a symbolic goal recognition problem defined in a planning language such as Planning Domain Definition Language (PDDL) (Ramırez and Geffner 2010; Freedman and Zilberstein 2017; Pereira, Oren, and Meneguzzi 2017; Harman, Chintamani, and Simoens 2018). By modifying the environment the intention recognition process can be made less challenging.

The term goal recognition design was recently coined by Keren, Gal, and Karpas (2014). In their approach the WCD of a problem is calculated by transforming the goal recognition problem into multiple planning problems containing pairs of goals. An optimal plan, with the longest possible non-distinctive prefix, to each pair of goals is searched for. The longest non-distinctive plan prefix, across all joint plans, is the WCD. To reduce WCD an increasing number of actions are removed until either the WCD is 0, or the search space has been exhausted in which case the best environment design discovered so far is returned. We will compare our solution to their pruned-reduce algorithm, and show that for navigation domains we have greatly reduced the time required to solve goal recognition design problems. Their approach has been extended for non-optimal agents (Keren, Gal, and Karpas 2015), and to determine where to place sensors within the environment (Keren, Gal, and Karpas 2016). In this paper we assume the agent/human is optimal and do not investigate sensor placement.

Wayllace et al. (2016; 2018) investigate goal recognition design involving stochastic action outcomes using Markov decision processes (MDPs). To calculate WCD the MDP for every goal is created, the states which are common to pairs of goals are discovered and the Bellman equation is used to calculate the cost of reaching a state. To reduce the WCD their algorithm removes a set of actions, checks that the optimal cost to reach a goal has not been affected, and calculates the WCD to find out if it has been reduced. Their approach creates a MDP multiple times for each of the goals, which results in large computational costs. In our approach a single model (Action Graph) is created, which contains the actions to reach all goals, moreover it is only created once.

Son et al. (2016) propose an approach based on Answer Set Programming (ASP) to reduce the computational cost, as an alternative to the Planning Domain Definition Language (PDDL) which the previously mentioned approaches use. Their results only show a maximum of two actions being removed, which greatly limits how much WCD can be reduced. In our approach we use PDDL, a popular domain-independent logic based formalism to represent planning problems, as we build on the work from (Harman, Chintamani, and Simoens 2018), which performed well (i.e. computational time and accuracy) on goal recognition problems.

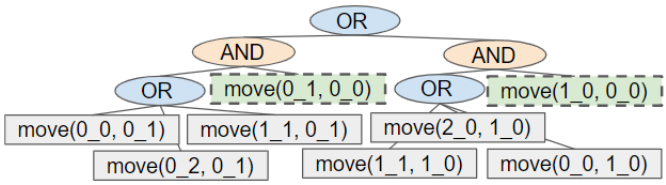
3 Background

Planning problems P are often specified in PDDL which consists of: a problem file containing a list of objects, the initial state and a goal state; and a domain file containing a set of action definitions. Formally they can be 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 $a_{pre} \subset F$ and effects $a_{eff} \subset F$ (Ramırez and Geffner 2010). A planner, for instance Fast Downward (FD) (Helmert 2006), solves a planning problem by finding the least costly sequence of actions leading from the initial state to the goal state.

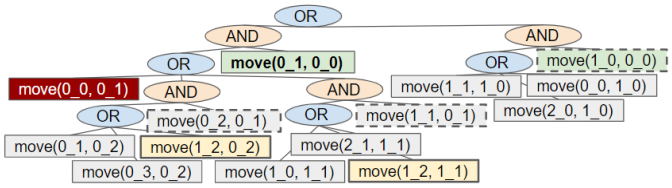
A goal recognition design problem can be defined as $\mathcal{D} = (F, I, A, \mathcal{G})$, where $\mathcal{G}$ is the set of all possible goals. These can also be defined in PDDL, i.e. a domain file, template file (problem without the goal statement) and hypothesis file containing the list of all possible goals. We aim to find a subset of actions $\hat{A} \in A$ which can be removed to reduce the WCD without increasing the cost of the optimal plan to any of the goals $\mathcal{G}$. In this paper the cost of a plan is its length.

4 Action Graphs for Goal Recognition Design

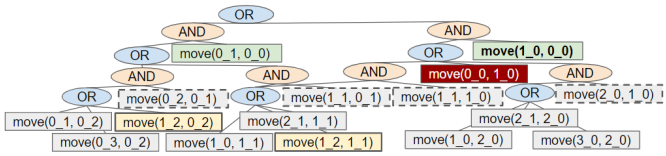
In this paper, we introduce a novel method for goal recognition design, which creates Action Graphs to model the constraints between actions. Action Graphs are produced from Domain Transition Graphs (DTGs), which to be generated require a planning problem (i.e. a PDDL problem and domain file). Our system begins by inserting a goal containing all of the possible goals $\mathcal{G}$ in an `or` statement into the template, to form a PDDL problem file. This problem along with its domain are transformed into a set of DTGs using the method developed by Helmert (2006) for FD. Every variable has its own DTG, which describes how the variable changes state. For instance, how a human's location changes from one position to the next. Subsequently, these DTGs are converted into an Action Graph.



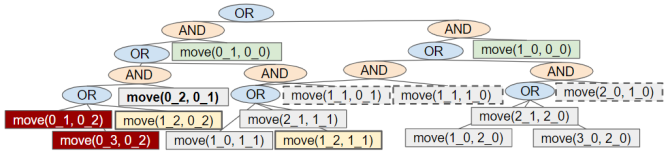
(a) The Action Graph creation starts by inserting the actions that result in the goal being reached (i.e. $\mathcal{A}_g = \{a \mid a_{eff} = g\}$), and their dependencies. The two goal actions, `move(0_1, 0_0)` and `move(1_0, 0_0)` are inserted into the BFS queue. If any of the goal actions are applicable to the initial state a threshold on the number of allowed steps away from the goal action is set to 0.



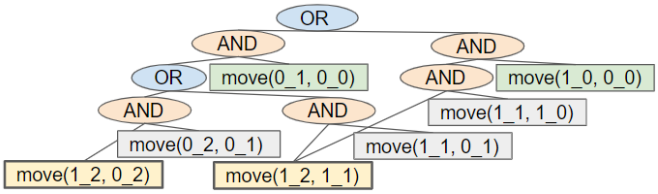
(b) `move(0_1, 0_0)` is popped from queue and the algorithm iterates over its dependencies, i.e. `move(0_0, 0_1)`, `move(0_2, 0_1)` and `move(1_1, 0_1)`. These actions also all have dependencies. All the dependencies of `move(0_0, 0_1)` have already been inserted into the graph, and connecting it to its dependencies would result in its dependencies being further from the goal, so the action is removed. For the remaining actions, i.e. `move(0_2, 0_1)` and `move(1_1, 0_1)`, their dependencies are successfully inserted into the Action Graph, and these two actions are appended to the queue.



(c) The next item in the queue, `move(1_0, 0_0)`, is processed in the same way. `move(1_1, 1_0)`'s dependencies already exist in the Action Graph, therefore it can be link to these without the need to create any action nodes.



(d) As a dependency of `move(0_2, 0_1)` is applicable to the initial state the threshold on the number of actions required to reach the goal is set to 2. Adding the dependencies for `move(0_1, 0_2)` and `move(0_3, 0_2)` would result in the threshold being breached, therefore these actions are removed. No further actions are inserted into the queue.



(e) The items in the queue continue to be processed as described in the previous steps until the queue is empty. This figure shows the resulting Action Graph, after a single goal has been inserted.

Figure 2: Example of the steps taken to create an Action Graph, when the initial state is `(human-at 1_2)` and the first goal is `(human-at 0_0)`. Actions which result in the goal state being reached are highlighted in green, yellow boxes with a thick border indicates actions applicable to the initial state, red indicates actions that are being removed, actions with a dashed border are currently in the BFS queue and the bold actions indicate the action which has just been popped from the queue.

After creating an Action Graph, actions belong to multiple goals are identified to formulate a set of non-distinctive plan prefixes. To reduce WCD, the non-distinctive plan prefixes are iterated through to discover which actions should be removed. This section first describes how Action Graphs are created, then how the non-distinctive plan prefixes are found and finally how WCD is reduced.

4.1 Creating an Action Graph

Action Graphs allow to model the order constraints, i.e. *dependencies*, between actions; they are similar to AND-OR trees as they contain `OR`, `AND` and leaf nodes. Leaf nodes are also referred to as action nodes, as each one is associated with an action. An `AND` node's children must be performed in order, and for `OR` nodes one or more of its children can be completed. Action, `OR` and `AND` nodes can have multiple parent nodes, as the Action Graph contains only one action node per action. Unless otherwise stated, the term parent(s) always refers to the direct parent(s) of a node.

An Action Graph is generated by performing a Breadth First Search (BFS) backwards from each goal $g \in \mathcal{G}$ to the initial state I. Figure 2 shows an example of the steps executed to insert all the plans for a single goal into an Action Graph. The graph is initialised with an `OR` node as the root and will receive a new child for every action whose effects result in a goal state being reached. Each goal $g \in \mathcal{G}$ is inserted in turn.

Actions which result in the goal state being reached ($\mathcal{A}_g = \{a \mid a_{eff} = g\}$) are found and inserted into to the graph along with their direct dependencies, as shown in Figure 2a. These goal actions and their dependencies are discovered by searching the DTGs. If any of the goal actions are

applicable to the initial state, $\mathcal{A}_I = \{a \mid a \in \mathcal{A}_g \wedge a_{pre} = I\}$, the maximum number of steps to reach the goal is 0; and no other actions, including any other goal actions (where $\hat{\mathcal{A}}_I = \{a \mid a_{pre} \neq I\}$), are inserted into graph. If no goal action is applicable to the initial state all goal actions' dependencies are inserted.

When an action has dependencies ($\hat{\mathcal{A}}_I = \{a \mid a_{pre} \neq I\}$) an AND node is inserted with the dependencies preceding the action as its children. As multiple actions (dependencies) can have the same effects, OR nodes are inserted to indicate the different paths. Once inserted into the graph, each goal action is appended onto a BFS queue.

For each action in the queue our algorithm iterates over their dependencies, to insert the dependencies' dependencies into the Action Graph. An example is provided in Figure 2b. We process actions and dependencies in this way because it is simpler to remove the action when none of its dependencies' dependencies can be inserted, e.g. if inserting them would result in a sub-optimal appearing in the graph. Dependencies are actions, thus in the subsequent text we refer to a dependency's dependencies as an action's dependencies.

If an action's dependencies have already been inserted into the graph, for the current goal, and connecting it to its dependencies would cause the dependencies to be further from the goal (i.e. create a cycle within the graph) the action is removed. After an action's dependencies have been inserted, the action is appended to the queue. This continues until an action that has no dependencies ($\mathcal{A}_I = \{a \mid a_{pre} = I\}$) is reached (Figure 2d).

When the initial state is reached a threshold on the number of allowed steps away from the goal is set, to prevent plans longer than the optimal from being inserted. Any actions with dependencies that would result in this maximum being breached are removed from the graph. If all an action's dependencies have been removed, then so too is the action itself. When an OR node contains only one child it is removed, i.e. the OR nodes parents become its child's parents. The process is completed when the BFS queue is empty. An example of an Action Graph produced by this process is shown in Figure 2e.

After all the plans to the first goal have been inserted, the same process is performed on the subsequent goal. The order the goals are processed does not affect the resulting Action Graph. When inserting the actions that lead to the subsequent goals, the algorithm also checks if an action was already inserted when processing a previous goal. If so, there is no need re-insert the action's dependencies; the threshold on the number of actions to reach the goal is set to the current action's distance from the goal plus the cost of its plan to reach the initial state, which is discovered by traversing the graph (see Algorithm 2).

4.2 Find all Non-Distinctive Plan Prefixes

Once an Action Graph has been created, the non-distinctive plan prefixes need to be discovered, and the WCD calculated, before iterating over the prefixes to remove the actions that result in a reduced WCD. In our approach finding non-distinctive plan prefixes involves two steps: 1) set which nodes in the graph belong to which goals and 2) find all the plans from the initial state up to and including each of the actions belonging to more than one goal, if an action is contained within another action's plan there is no need to also find its plan. This sections describes these two steps in more detail.

Set Which Nodes Belong to Which Goals Each goal action $\mathcal{A}_g = \{a \mid a_{pre} = g \in \mathcal{G}\}$, which requires dependencies, has an AND node as its parent, all the AND node's children including non-direct children (e.g. children's children) belong to the same goal as the goal action. Therefore, by performing a depth first tree traversal starting from the AND node, each node is provided with a list of goals it belongs to.

Algorithm 1 Get non-distinctive plans from Action Graph

```
1: function GET_NON_DISTINCTIVE_PREFIXES(graph)
2:     non_distinctive_prefixes = ∅
3:     for a ∈ graph.get_all_actions do
4:         if a only belongs to 1 goal or a ∈ non_distinctive_prefixes then
5:             continue
6:         end if
7:         plan = GET_PLAN(a)                          ▷ gets any plan
8:         for p ∈ non_distinctive_prefixes do
9:             if p.last_action = a then
10:                non_distinctive_prefixes.rm(p)
11:                break
12:            end if
13:        end for
14:    end for
15:    sort(non_distinctive_prefixes)                  ▷ Longest 1st
16:    WCD = non_distinctive_prefixes[0].size
17:    return non_distinctive_prefixes
18: end function
```

Algorithm 2 Get any plan containing the given action. Returned plan ends with the action itself.

```
1: function GET_PLAN(a)
2:     plan = ∅
3:     if a.parents.size = 1 and a.parent is AND node then
4:         GET_PLAN_FOR_NODE(a.parent, plan)
5:     end if
6:     plan.append(a)
7:     return plan
8: end function
9: function GET_PLAN_FOR_NODE(node, plan)
10:    child = node.children[0]
11:    if child is AND node then
12:        plan.append(GET_PLAN(child.children[1]))
13:    else if child is OR node then
14:        plan.append(GET_PLAN_FOR_NODE(child))
15:    else                                            ▷ node is an action node
16:        plan.append(child)
17:    end if
18: end function
```

Find Non-Distinctive Plan Prefixes To find the non-distinctive plan prefixes, Algorithm 1 iterates over all the actions which belong to more than one goal, and adds their

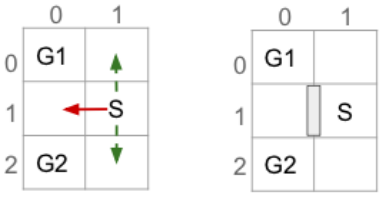
plans to a list of non-distinctive plan prefixes. We use the term *action's plan* to refer to a plan from the initial state which ends with that action. To prevent multiple (sub-)plans being repeated, if the action has already been inserted into one of the non-distinctive plan prefixes it is ignored, and the algorithm removes plans from the list which end with an action in a newly found plan (lines 8-13). The WCD is the length of the longest non-distinctive plan prefix.

An action's plan is discovered by traversing the Action Graph depth first (Algorithm 2). When an `OR` node is encountered a single child is selected. It does not matter which one of an action's plans are discovered, as those actions not traversed will also appear, within a different plan, in the list of non-distinctive plans. Thus, will still be processed during action removal to reduce WCD (see Section 4.3). As only optimal plans have been inserted into the Action Graph, this does not affect calculating WCD.

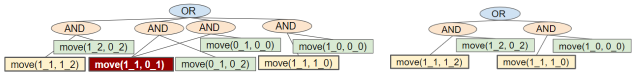
4.3 Action Removal to Reduce WCD

We aim to reduce the WCD without increasing the cost of the optimal plan to any of the goals, by preventing actions from being performed. This section describes how the actions to remove are selected. In Figures 3, 4 and 5 we provide some simple examples to illustrate how our approach works; our approach is applicable to environments of any size with any number of goals.

The list of non-distinctive plan prefixes is sorted, most costly first; so the worst is processed first. In turn, each prefix is removed from the list, and its actions iterated over to find one in which all the goals it belongs to have an alternative action.



(a) The image on the left shows an example of an environment with a single non-distinctive plan prefix containing 1 action (shown as solid red arrow), for which each goal has an alternative action (dashed green arrows). As each goal has an alternative, the non-distinctive action can be removed, resulting in a WCD of 0 (right).



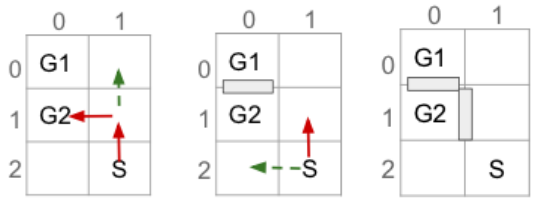
(b) The action graph for the example environments. The action which is removed is shown highlighted in the image on the left (white text with red background). The Action Graph after the WCD has been reduced is shown on the right.

Figure 3: Example goal recognition design problem, in which both goals have an alternative to the plan(s) containing the non-distinctive prefix.

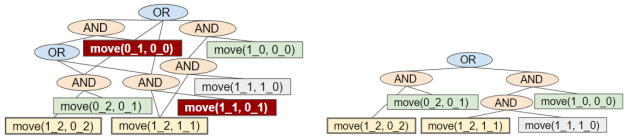
A goal has an alternative action, if the action (or if the action has dependencies its single `AND` parent) has an `OR` node as a direct parent, and another one of the `OR` node's children belong to that goal. An example is provided in Figure 3. Moreover, the alternative cannot belong to any of the other goals the (non-distinctive) action belongs to, e.g. the first action in Figure 4a (left image). If all the goals, with a plan containing the action from the non-distinctive plan prefix, have an alternative action, the action is removed.

After checking all actions in a non-distinctive plan prefix, if only a subset of the goals have an alternative action, the action(s) directly after the non-distinctive prefix in their plans are removed. An example is shown in Figure 4. If the goals have an alternative action but their alternatives are the same, the next (distinctive) action(s) for one of the goals is removed (see Figures 5c-d). Our action removal method always checks that the action removed will not interfere with any of the other goals, which do not have an alternative.

Once an action has been removed, which nodes belong to which goal is re-evaluated (see Section 4.2) and the actions in the non-distinctive plan prefix, prior to any removed action, are checked to see if they should be inserted into the list of non-distinctive plan prefixes (e.g. Figure 4a). The last action in the plan prefix will not be processed again, if it is still a non-distinctive action then the WCD will not be reduced to 0. An example of an environment in which the WCD cannot be reduced to 0, and the steps our algorithm performs, is shown in Figure 5.

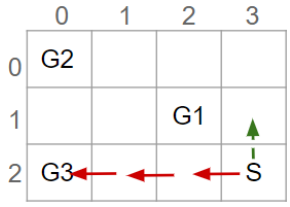


(a) Both actions have an alternative to the first action in the worst non-distinctive plan prefix. The alternative action, `move(1_2, 0_2)`, also belongs to both goals and is thus only taken into consideration as a last resort. G1 has an alternative action to the second action, whereas G2 does not. So, the next action in G1's plan, which contains the non-distinctive plan prefix, is removed. After removing an action (middle image), `move(1_2, 1_2)` is still a non-distinctive action, thus is evaluated on the next iteration. This time G2 is the only goal with an alternative action. The environment after all non-distinctive plan prefixes have be evaluated is shown in the image on the right.

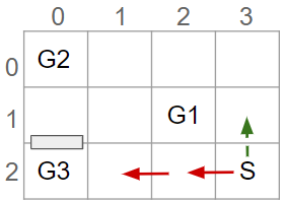


(b) Initial Action Graph shown on the left. The Action Graph after the action removal process has completed is shown on the right

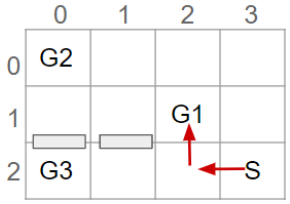
Figure 4: Example of WCD reduction, when only a subset of the goals have an alternative action, to an action in the non-distinctive plan prefix.



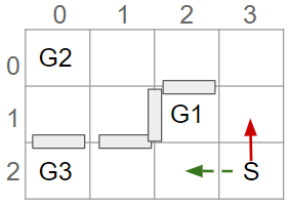
(a) Initial environment. Only G2 has an alternative action, for any action in the non-distinctive plan prefix, so the next action in G2's plan which contains the non-distinctive prefix is removed.



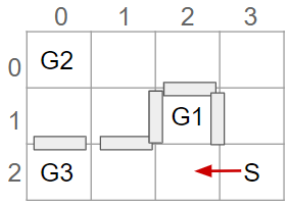
(b) After the first action has been removed the second to last action in the original non-distinctive plan prefix is still a non-distinctive action, thus is evaluated in the next iteration.



(c) After the second action has been removed the longest non-distinctive plan prefix belongs to G1 and G2. Neither G1 nor G2 have an alternative action, which does not belong to the other. Therefore, the next actions(s) in only 1 of the goal's plans are removed. In this situation, as G1 has no further actions, G2's next actions are removed.



(d) Two actions have been removed, since there were two possible next actions to reach G1 (discovered by detecting an OR node in the Action Graph).



(e) G2 is now distinctive, but there still exists a non-distinctive plan prefix for G1 and G3.

Figure 5: Example of a 3 goal environment, in which WCD cannot be reduced to 0. In each environment design a longest non-distinctive plan is indicated with red arrows.

5 Preliminary Results

Through experiments we aim to show: 1) how well our approach scales as the number of goals and size of the environment is increased, 2) how much the WCD is reduced, and 3) how many actions are removed. We compare our Action Graph approach to the pruned-reduce method by Keren, Gal, and Karpas (2014), by running both approaches on a dataset we created containing grid-based navigation goal recognition design problems with randomly selected initial and goal locations. This section presents a description of the experiment setup followed by a discussion on the results.

5.1 Setup

To perform experiments we created two grid-based navigation datasets containing goal recognition design problems. The first contains problems with an 8 by 8 grid and a varying number of goals. For each number of goals 8 problems were generated by randomly selecting the human's start location and the goal locations. In total the dataset contains 112 problems.

The second dataset consists of problems with differing grid sizes, for all problems both the width and the height are equal. For each grid size 8 problems with a random start location and three random goals were selected. In total this dataset contains 56 goal recognition design problems.

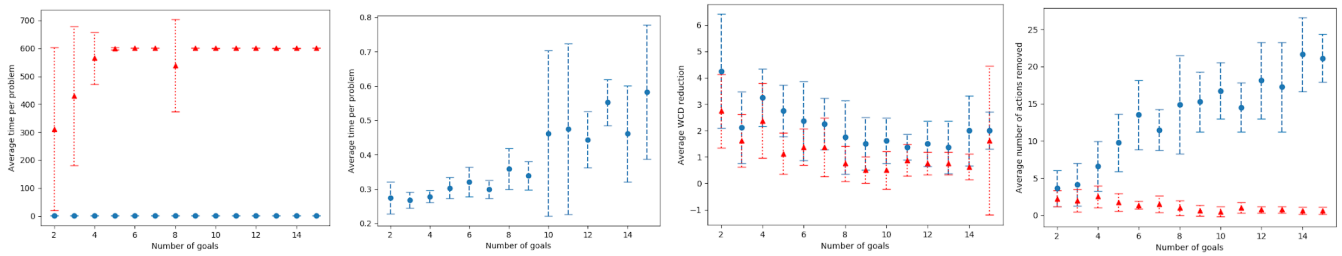
Experiments were ran on a server with 16GB of RAM and a Intel Xeon 3.10Ghz processor. A timeout of 10 minutes per goal recognition design problem was set for all experiments. The whole process, starting from converting the PDDL into SAS+ (then creating the DTGs and building an Action Graph) is included in the run-times for our approach.

5.2 Results and Discussion

The average time, WCD reduction and number of actions removed for an increasing number of goals and an increase grid size are shown in Figures 6 and 7 respectively. Both experiment results follow a similar trend.

For each of the problems within the varying number of goals dataset our Action Graph approach took less than 1 second to perform goal recognition design. Whereas, pruned-reduced hit the timeout for the majority of problems (Figure 6a). The standard deviation of the execution time varies greatly between different problems (Figure 6b), as for some problems removing a small number of actions reduces WCD to 0 whereas for others WCD can only be partially reduced, and the optimal plans within the different problems differ in length (longer plans take more time to find and iterate over).

Our approach has managed to reduce WCD more than pruned-reduce but has removed more actions. Pruned-reduced attempts to remove an increasing number of actions thus, as shown in Figure 6d, did not attempt to remove many actions before the timeout was reached. This has resulted in pruned-reduced not reducing the WCD as much as our approach (Figure 6c). Having said that, our approach occasionally removes more actions than is necessary, e.g. Fig-



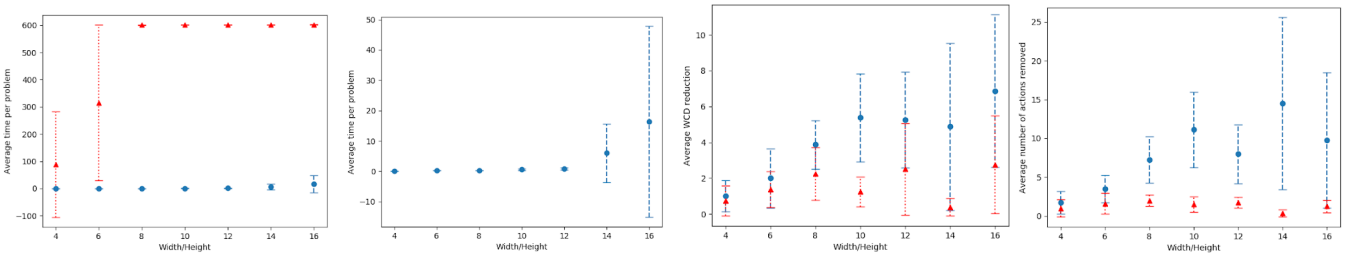
(a) Average time per goal recognition design problem.

(b) Average time per goal recognition design problem, for just our approach.

(c) Average WCD reduction.

(d) Average number of actions that were removed.

Figure 6: Results for an increasing number of goals. The results of our Action graph approach is indicated by blue circles, pruned-reduce (Keren, Gal, and Karpas 2014) is shown with red triangles. All times are given in seconds.



(a) Average time per goal recognition design problem.

(b) Average time per goal recognition design problem, for just our approach.

(c) Average WCD reduction.

(d) Average number of actions that were removed.

Figure 7: Results for an increasing grid size. The results of our Action graph approach is indicated by blue circles, pruned reduce (Keren, Gal, and Karpas 2014) is shown with red triangles. All times are given in seconds

ure 5, therefore in future work we will investigate different methods of selecting which actions to remove, for instance by reversing the order the non-distinctive actions are iterated over. The same trends, as increasing the number of goals, are observed for an increasing grid size (Figure 7).

These experiments have proven that our Action Graph approach is more scalable than a current state-of-the-art approach to goal recognition design for grid-based navigation domains. Our approach has managed to reduce the WCD of 168 grid-based navigation environments, with various numbers of goals and grid sizes, from an average of 6.29 actions to 3.46 actions, in an average of 1.42 seconds per problem.

6 Conclusion and Future Work

In this paper we have described an early version of our work on goal recognition design using Action Graphs for grid-based navigation domains. Action Graphs are generated by performing an adapted BFS backwards from each goal state to the initial world state. Subsequently, the graph is traversed to discover all the non-distinctive plan prefixes and to calculate the WCD. Finally, the non-distinctive plan prefixes are iterated over to search for actions that can be removed, to reduce the WCD. Our initial experiments show promising results, and we are currently looking at how to improve and extend this work.

There are environments in which the overall WCD cannot be reduced, however the distinctiveness between certain goals can be. Therefore, in the future we will introduce a new metric that better encapsulates the how distinctive each of the goals are.

In realistic human occupied environments, the only feasible domain, we can think of, in which actions should be completely removed is navigation. For other domains, we will investigate how the state of the environment can be modified to reduce WCD. For instance, which cupboards items should be placed in within a smart kitchen or smart factory environment.

We will also investigate extracting which actions need to be sensed from the Action Graph. Moreover, goal recognition experiments will be performed in a real world smart home to determine how much the designed environment has improved the accuracy of recognising a human's goal.

7 Acknowledgements

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

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 *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 918–924.

Freedman, R. G., and Zilberstein, S. 2017. Integration of planning with recognition for responsive interaction using classical planners. In *Thirty-First AAAI Conference on Artificial Intelligence*, 4581–4588.

Goldman, R. P.; Geib, C. W.; and Miller, C. A. 1999. A new model of plan recognition. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, 245–254.

Harman, H.; Chintamani, K.; and Simoens, P. 2018. Action trees for scalable goal recognition in robotic applications. In *the 6th Workshop on Planning and Robotics (PlanRob)*, 90–94.

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 *Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2016)*, 1489–1496. IEEE.

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal recognition design. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling*, ICAPS, 154–162. AAAI Press.

Keren, S.; Gal, A.; and Karpas, E. 2015. Goal recognition design for non-optimal agents. In *AAAI Conference on Artificial Intelligence*, 3298–3304.

Keren, S.; Gal, A.; and Karpas, E. 2016. Goal recognition design with non-observable actions. In *AAAI Conference on Artificial Intelligence*, 3152–3158.

Keren, S.; Gal, A.; and Karpas, E. 2018. Strong stubborn sets for efficient goal recognition design. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 141–149.

Pereira, R. F.; Oren, N.; and Meneguzzi, F. 2017. Landmark-based heuristics for goal recognition. In *Thirty-First AAAI Conference 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 Conference 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.

Son, T. C.; Sabuncu, O.; Schulz-Hanke, C.; Schaub, T.; and Yeoh, W. 2016. Solving goal recognition design using asp. In *AAAI Conference on Artificial Intelligence*, 3181–3187.

Wayllace, C.; Hou, P.; Yeoh, W.; and Son, T. C. 2016. Goal recognition design with stochastic agent action outcomes. In *the International Joint Conference on Artificial Intelligence (IJCAI)*.

Wayllace, C.; Keren, S.; Yeoh, W.; Gal, A.; and Karpas, E. 2018. Accounting for partial observability in stochastic goal recognition design: Messing with the marauders map. *the 10th Workshop on Heuristics and Search for Domain-Independent Planning (HSDIP)* 33.

Zhuo, H. H., and Li, L. 2011. Multi-agent plan recognition with partial team traces and plan libraries. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI 2011)*, volume 22, 484.