Planning Human-Robot Interaction Tasks Using Graph Models

L.J. Manso[1], P. Bustos[1] and R. Alami[2], G. Milliez[2], and P. Núñez[1]

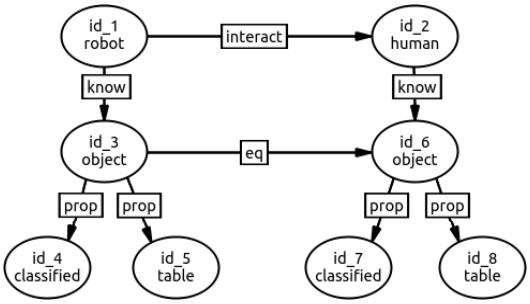
[1] Universidad de Extremadura,
Av. de la Universidad sn, 10071, Cáceres, Extremadura,
lmanso@unex.es
[2] Laboratoire d'Analyse et d'Architecture des Systèmes, CNRS,
7 Avenue du Colonel Roche, 31400, Toulouse, France,

Abstract. Human-robot interaction is a complex field of robotics in which robots are required to deal with different challenging issues. Among other skills, HRI-capable robots need to generate plans taking humans into account. To achieve this robots require sufficiently efficient data structures and rich information about their environment as well as about humans and their abilities. An additional requirement when robots are supposed to actively find and classify objects is the capability of reasoning about the creation and retyping of the symbols corresponding to the objects as a result of the actions of their plans. This paper describes how these requirements can be met using a combination of dynamic graph-like world models and a planning system based on graph-rewriting rules. To demonstrate how the approach can be applied, the paper builds upon a robot butler use-case, describing how its world model is structured and its most relevant planning rules. Qualitative and quantitative experimental results are also provided.

Keywords: human-robot interaction, graph models, theory of mind

1 INTRODUCTION

The Human-Robot Interaction (HRI) community aims for robots capable of successful interaction with human beings. These interactions are generally complex: robots must usually take into account not only their knowledge about the objects in the environment but also knowledge about the humans they interact with and the knowledge they may have. It must be especially noted that for successful interaction robots must be endowed with a *theory of mind* [2], enabling them to understand how the knowledge of humans is updated as new events occur. For example, when a human commands a robot to bring an object, the robot should not necessarily stop when the object is close to the human but when it is close *and* the human realizes that. To infer the knowledge of the humans a robot interacts with it must be able to take their perspective to compute and keep track of the objects they see [15]. Indeed, some goals must be planned taking into account the beliefs of the robots on the knowledge of the humans instead of the



(a) Model example

	value
x	900
y	-487
z	1605
ry	0
type	round
size	798
height	804

(b) Attribute map.

Fig. 1. a) depicts the model of the robotic butler after detecting a human and a table. The model also shows that the robot believes that the human knows about the table (more information in Sec. 3). The labels of the edges indicate: *interact*, that the robot interacts with another agent; *eq*, correspondence between the knowledge a robot holds on an object and the one it estimates is held by another agent; *know* is used to denote the owner of objects; and *prop* is used to add properties to objects. **b)** provides an example of the attribute map a table object could have.

direct knowledge of the robots. Moreover, when performing a collaborative task involving a human, having knowledge on human's knowledge allows the robot to communicate and synchronize the information required to execute the plan.

Reasoning about the perceptual state of humans as a means to achieve the goals of the robots requires sophisticated reasoning rules and appropriate data structures [18]. To satisfy their information requirements efficiently robots need metric and symbolic information. Metric information is necessary for low level reasoning, generally about statements regarding the physical world, while symbolic information is required to work with higher-level information. Symbolic facts are generally stored as sets of predicates, which are equivalent to labeled hyper-graphs (*i.e.,* hyper-graphs whose edges are labeled). For example, a predicate (*travel A B t*) can be seen as an edge labeled as *travel* associated to the (A, B, t) vertex tuple. However, hybrid models combining metric and symbolic properties in the same structure are gaining importance in the last years, graph models and equivalents in particular [3, 4, 11, 16].

The work presented in [4] uses models particularly similar to the ones proposed here or in previous work by the authors [16] as a means to integrate perception, control, planning and coordination. The present paper also suggests using hybrid models, specifically, an extension of the concept of *attributed graph* previously presented in [16], where an attributed graph G^* is defined as a tuple of two sets (V^*, E^*), being V^* a set of vertexes –symbols– with a type and an identifier (that can also be attributed with additional metric properties), and E^* a set of labeled edges. Each vertex in V^* contains a map of properties that can be used to store the metric attributes of the symbol they represent. Edges in E^* are labeled with a string used to identify the nature of the edge (see figure 1). While the theoretical expressive power of this data structure is not higher than

the one of a regular graph, we suggest that they make understanding models easier for humans and, in contrast to the hypergraphs used in the *Planning Domain Definition Language* (PDDL) [17] their graphical representation can be easily automated [12].

The contributions of the paper are threefold: **a)** we demonstrate how graph models can be applied to HRI domains, supporting to some extent a theory of mind for robots; **b)** we present a new visual planning domain definition language named *Active Graph Grammar Language* (AGGL) that is designed to work with dynamic graph models; and **c)** we evaluate the performance of a novel AGGL-based planner named *AGGLPlanner* in HRI domains.

The remainder of the paper is organized as follows. Section 2 introduces the AGGL domain specification language. Section 3 describes how graph models and graph-rewriting rules can be used to represent the knowledge and the domain of a robotic butler endowed with some features of a theory of mind. Sections 4 and 5 present the experimental results obtained and the conclusions that can be extracted from the paper, respectively.

2 THE ACTIVE GRAPH GRAMMAR LANGUAGE

Regarding the planning rules that robots can use when looking for valid plans to reach their goals, there is a limitation that is not usually met in regular planning systems (in particular those based on PDDL). Such limitation is that planning languages (and therefore their corresponding planners) do not natively support creating, deleting nor retyping symbols as a consequence of the execution of planning actions, that is, the number of symbols and their type is fixed. This makes harder to generate domains in which the robot is able to discover, forget or reconsider the type of objects, which is an essential requirement for different tasks, including planning perceptive tasks [6, 14]. While there are workarounds to overcome the limitation, they make domain description files harder to read (especially when the actions are complex) and decrease the efficiency of planners (see [14] and section 4). Additionally, the efficiency of state-of-the-art PDDL planners heavily decreases with the number of parameters of the actions. Since HRI planning actions are often complex, this scalability issue represents a noticeable problem. On the other hand, the efficiency loss can be overcome splitting complex planning actions in several *virtual actions* with fewer parameters but, again, this solution makes domains harder to read and no longer reflect the actual domain. The Active Graph Grammar Language (AGGL) was created from the motivation that only computers should deal with this kind of optimizations and that the work of domain designers should be made as easy as possible.

AGGL can be seen as an extension of PDDL [17] with two advantages: one regarding readability and another regarding the power of the language. One of the drawbacks of PDDL and most textual planning languages is that complex actions such as the ones needed for HRI are hard to write and read. The readability of the rules decreases with the number of symbols and predicates in the actions, and when these numbers are high they are difficult to understand.

Listing 1.1. This code is the PDDL-equivalent of the graphical rule described in Fig. 2. This kind of textual description is harder to understand and requires more training. Predicates $ISxxx$ are used to allow dynamic symbol typing. Predicates $firstunknown$ and $vOrd$ are used to maintain a list of symbols available to use (see [14]). The rest of the predicates have the same meaning as the edges in Fig. 2.

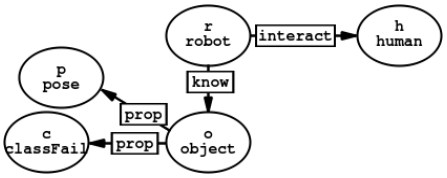
```
(:action tellHumanAboutUnknownObject
 :parameters ( ?p ?c ?r ?vo ?h ?lstAux ?aux0 ?aux1 ?aux2
  ?aux3 ?aux4 )
 :precondition (and (ISpose ?p) (ISclassFail ?c) (ISrobot ?r)
  (ISobject ?vo) (IShuman ?h) (firstunknown ?aux4)
  (vOrd ?aux4 ?aux3) (vOrd ?aux3 ?aux2) (vOrd ?aux2 ?aux1)
  (vOrd ?aux1 ?aux0) (vOrd ?aux0 ?lstAux) (know ?r ?vo)
  (interact ?r ?h) (prop ?vo ?p) (prop ?vo ?c) )
 :effect (and (not(firstunknown ?aux4))
  (firstunknown ?lstAux) (ISreach ?aux4) (ISsee ?aux3)
  (ISpose ?aux2) (ISnoClass ?aux1) (ISobject ?aux0)
  (prop ?aux0 ?aux2) (prop ?aux0 ?aux3) (eq ?vo ?aux0)
  (prop ?aux0 ?aux4) (know ?h ?aux0) (prop ?aux0 ?aux1) )
)
```

Readability declines even more when users have to implement the workaround introduced in [14] to get rid of the limitation of PDDL regarding symbol creation and deletion (code listing 1.1 provides an example of this issue).

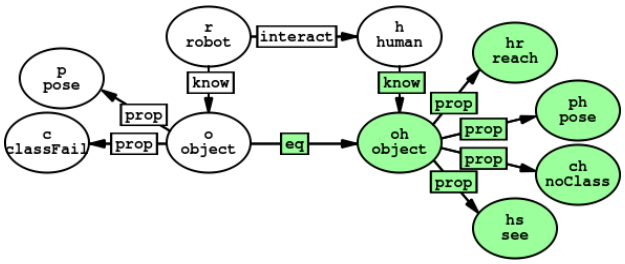
To solve these problems AGGL allows users to describe planning actions visually as *graph grammar rules* [16] (see Fig. 2). They are defined as pattern pairs, in the same way as string grammar rules: each rule states that the pattern in the left-hand side can be replaced with the pattern in the right-hand side. Each node in these rules has two strings, the upper one corresponds to the identifier of a parameter and the one in the bottom to its type. New symbols (those which only appear in the right-hand side pattern) are automatically highlighted in green by the editing tool, symbols which are going to be deleted (those in the left-hand side pattern only) are highlighted in red, and the symbols whose type is modified by the rule are highlighted in gray. As can be appreciated comparing Fig. 2 and List. 1.1, AGGL helps improving the readability of domain descriptions.

Since there are some features that can not be easily expressed graphically (*e.g.,* quantifiers), AGGL allows users to write additional precondition and effect clauses. These clauses extend the PDDL semantics by including three new capabilities necessary for some perception and HRI-related tasks which are not present even in the newest extension of PDDL:

- **retype:** allows to change the type of existing symbols *e.g., (retype name new_type).*
- **create:** is used to create new symbols and specify their type *e.g., (create name type).*



(a) Left Hand Side (LHS)



(b) Right Hand Side (RHS)

Fig. 2. An AGGL planning action used in a robot butler to allow the robot share information with humans. In particular, this rule is associated to the fact that robots can show humans objects they do not know, for example, to ask them about the objects later. The symbol r denotes a robot, h a human, o an object that the robot knows about, and p and o are properties of the object indicating that its pose is known but its classification is not. The execution of the action implies that five new symbols would be created: an object *oh* associated to the human which refers to the same object that o does but with different properties.

– **delete:** allows to destroy a symbol given its name *e.g., (delete name).*

Previous work by the authors [16] proposed a translator from an early version of AGGL to PDDL in order to avoid writing textual planning rules (such as the one in List. 1.1) while allowing creating and deleting objects dynamically [16]. However, this approach has efficiency issues. The most efficient domain-independent planners perform an initial instantiation phase (known as *grounding*) which generates a full instantiation of the possible actions as a prelude to the search process. This phase is used to compute efficient heuristics to guide the search process, but often leads to a combinatorial explosion if the problem has a considerable size or if the actions have a large number of parameters [8], specially if the domains require (simulating) symbol creation as it is the case of active perception domains. Some works such as [1] focus on minimizing the impact of the instantiation phase by automatically splitting the actions of the domains into several (virtual) actions with fewer parameters (*i.e.,* the complexity of the instantiation phase for an action grows exponentially with the number of its parameters). Although it is a promising research direction, the PDDL subset supported by [1] does not include *forall* nor *exists* PDDL clauses, which are necessary for most real-life HRI domains. Other works are focused towards reducing the complexity

of the instantiation phase [7], however none of these improvements are sufficient for the problems considered here. Thus, PDDL planners are –to the date– particularly unsuitable for perception and HRI tasks for two key reasons: **a)** the emulation of variable-sized worlds is inefficient; and **b)** HRI-related planning actions are complex in terms of the number of parameters used, which makes the instantiation phase grow out of control. The second issue could be circumvented breaking up actions manually, but this re-formulation is usually hard to perform and has an undesirable impact on readability and maintainability: this optimizations should only be performed automatically. The first issue has no known solution.

In order to improve efficiency in scenarios of dynamic world sizes, where it is necessary to work with complex rules, a new planner working natively with AGGL was created. The planner, named *AGGLPlanner*, avoids the initial instantiation phase and performs an A^* search using the number of conditions met as heuristic, checking all preconditions before instantiating new possible world states. Avoiding the instantiation phase sacrifices the computation of efficient heuristics that might speed up the search process, but it allows us to natively support symbol creation, deletion and retyping, and prevents the otherwise inevitable combinatorial explosion. This makes the planner particularly useful in domains with large number of symbols and parameters, where the complexity lies more in the domain than in the length of the plan. Moreover, due to the nature of the search process it was possible to implement macro operators, which can be introduced manually [13] or automatically using machine learning techniques [5]. Additionally, *AGGLPlanner* was made anytime (*i.e.,* the user can stop the planner at any time and still get the best plan among the valid ones found, if any), which might be important in some scenarios. On the other hand, due to the lack of sophisticated heuristics, the planner is not competitive versus state of the art planners if the domain works with fixed-sized world models and simple actions. Additional technical information regarding the planner is out of the scope of the paper.

3 GRAPH MODELS FOR A ROBOTIC BUTLER

In this section and for the remainder of the paper we build on the example of a robotic butler to show how different planning-related HRI phenomena can be implemented using graph models and planning rules that can work with dynamic world models. Only the most relevant of rules used in our experiment will be described, since showing all of them would require too much space and it would not provide valuable additional information. In particular we describe three rules which allow the robot to find new objects, model unknown objects and find human-aware plans to serve coffee.

The world model used in this work contains a symbol for each of the physical agents (*i.e.,* the robot and the humans) and an edge labeled as *interact* that links the symbol of the robot to those of the humans the robot interacts with. To make the robot able to plan complex tasks, taking into account not only its direct

knowledge but also its beliefs regarding the humans, the model holds duplicated information for every object, containing potentially different data. This data depends on the robot's beliefs and relationships with the objects and the ones of the humans (as perceived or expected by the robot). This is particularly important because a considerable amount of goals are given –or should be given– from a third-party perspective. For example, if a robot is given the goal "bring me the ball", the goal of the robot should not be just to take the ball close to the human, but also ensure the human knows the ball is actually close to him (bringing the ball to a human is generally useless until the human realizes its new position). Therefore, for every object known by the robot, it creates a symbol linked to the symbol of the robot (depicting that the robot is aware of the object) and another for every human that the robot believes is aware of the same object. Each *object* symbol is connected to other symbols whose types encode different properties of the objects from the point of view of the agent they are associated to (*e.g.,* an object might be reachable from the point of view of the robot but not from the one of the human). Finally, objects that are located in other objects are linked to their container with an edge labeled as *in*. Fig. 1 depicts a small example of the models used.

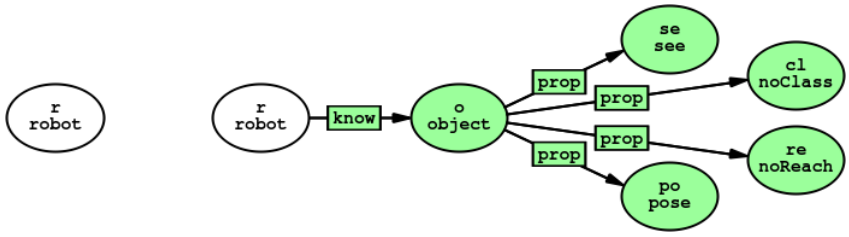
The symbols encoding the properties of the objects in this work can be: **a)** related to its **classification status**: that is, *class*, *noClass* or *classFail* (classFail meaning that the robot failed to classify the object); **b)** a **categorization** symbol (for the experiment we consider *mug*, *table*, *coffeePot* and *milkPot* symbols) and **c)** other symbols related to the **information or capabilities of the agents** in relation to the object: *see*/*notSee*, *pose*/*notPose* and *reach*/*notReach*. To acquire perception-related facts (*e.g.,* whether or not a human sees or can reach an object) we assume the use of a third-party geometrical perspective-taking modules such as the ones proposed in [20].

The rest of this section describes three of the most relevant planning rules used in the robotic butler experiment, specifying how can the robot: find and classify new objects, tell humans about objects it knows, and ask for help to classify objects (taking into account that humans have some capabilities that robots do not).

3.1 Object discovery

The rule for object discovery is the easiest to understand. It creates an *object* symbol whose properties are: *see* (it is assumed to be seen because it is applied when the robot discovers an object visually), *pose* (because it is seen its position is known), *noReach* (if the object is actually reachable when discovered, the robot can later update the property), and *noClass* (objects are initially assumed to be unknown until they are inspected). The new object is linked to the symbol of the robot. The rule, named *findObjectVisually*, is shown in Fig. 3.

Reachability is a precondition to execute the actions classifying objects, as a means for active perception. This way the robot is enforced to approach to



(a) LHS (b) RHS

Fig. 3. Graphical representation of rule "findObjectVisually".

the objects it intends to classify, to obtain a close view (the specification of the complete set of planning rules can be obtained from the web page of the paper[3]).

3.2 Updating the properties of objects

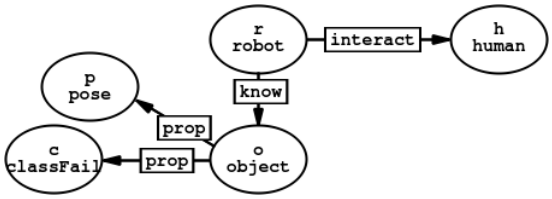
Updating the properties of the objects can be a planned or an exogenous event, and is reflected in the model held by the robot by changing the type of the corresponding property. For example, when the robot realizes or plans to start seeing an object, the event is reflected in the graph by changing the corresponding *notSee* symbol to *see*. The same applies for the robot and humans, and for other properties.

3.3 Cooperative perception

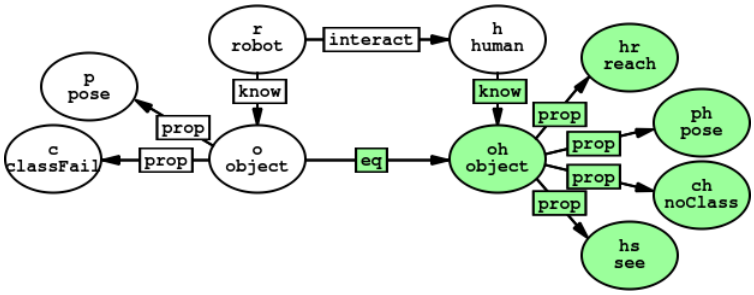
There are many different ways robots and humans can cooperate in perception tasks. They can show each other objects they do not know or ask for the position of an object both know. In order to be able to achieve these cooperative actions, the individual beliefs and capacities of the agents (*i.e.,* humans or robots) must be represented. Among all the possibilities, the paper provides two examples related to the experiment described in section 4.2: how can a robot tell a human about an object it is unable to classify (Fig. 4, named *showUnknownObject*), and how can the robot ask a human for the type of an unknown object (Fig. 5, named *humanTellsItsMilk*).

The effect of rule "showUnknownObject" (Fig. 4) is that the robot assumes the human knows the object. Therefore, the robot inserts the corresponding new symbols (shown in green) and attaches them to the human symbol. The effect of the rule "humanTellsItsMilk" (Fig. 5) is that the object is considered classified from the point of view of the robot (changing the type of the parameter f, shown in gray) and that a new type-symbol *milkPot* is created and associated to the object. These rules are used in combination with another rule (not shown due to space limitations) in which the human classifies the initially unknown object as a milk pot.

[3] Web page providing additional resource files related to this paper: http://ljmanso.com/REACTS15



(a) LHS



(b) RHS

Fig. 4. Graphical representation of rule "showUnknownObject".

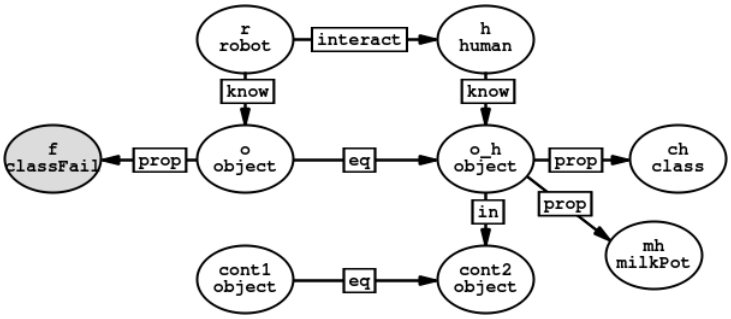
4 EXPERIMENTAL RESULTS

Two different kind of results are provided. First, the comparison on how the number of symbols available to create affects the execution time in different planners, and second, the definition and execution of the experiment.

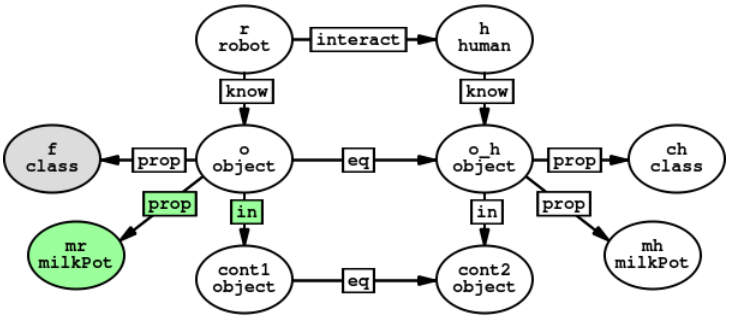
4.1 Impact of the number of available symbols

For this experiment the system was given a very simple task: finding a new object. The experiment compared the execution time of AGGLPlanner with two well-known planners: Metric-FF [10] (last version, 2.1) and FastDownward [9] (several versions, obtaining similar results[4]). The planning problem (which requires creating at least five new symbols) was given to Metric-FF, FastDownward and AGGLPlanner. Metric-FF could not be successfully used because it crashed during the instantiation process (it consumed all the computer's memory). FastDownward could solve the problem using a pool of new symbols from five (the minimum necessary to solve the problem) to eight, but the execution time grew exponentially with the number of such symbols (handling more memory caused the same problem that Metric-FF had). Since AGGLPlanner avoids grounding, it was not affected by the number of potential new symbols. The execution time of the experiments is shown in Fig. 6 (in logarithmic scale).

[4] Three versions of the FastDownward software repository fetched between June 2013 and January 2015 were tested, obtaining similar results.



(a) LHS



(b) RHS

Fig. 5. Graphical representation of rule "humanTellsItsMilk".

The results, clearly favorable for AGGLPlanner, were obtained because it was specially designed with the possibility of creating and deleting symbols in mind, an objective that was not considered in the design of FastDownward, Metric-FF or any other PDDL planner. Almost any other task not requiring symbol creation nor complex actions would run faster in FastDownward, especially when the search process requires complex heuristics. However, these preconditions are seldom found in human-robot interaction domains in which the robot has to reason about about humans' beliefs and potentially find new objects.

The files used to run the experiment can be obtained from the web page created for this paper (see footnote 3).

4.2 Serve coffee experiment

The experiment takes place in a kitchen environment, where a human and a robot are interacting and at some point the human tells the robot to serve a cup of coffee. The experiment assumes that the robot is endowed with third-party modules to assess what the human knows, reaches and sees (*e.g.,* [20]), that the robot understands the order "serve me coffee" as "put milk, coffee and a mug in a table reachable by the human", and that the human and the robot are

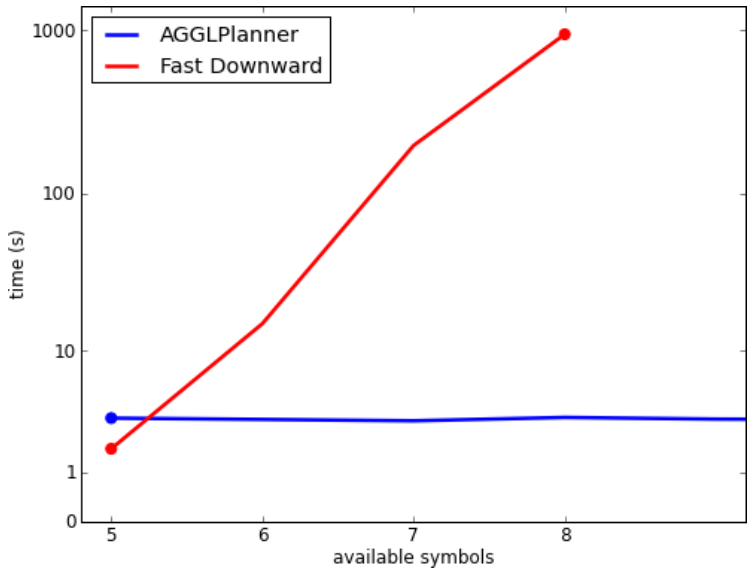


Fig. 6. Execution time of AGGLPlanner and FastDownward for the task of finding a new object. It must be noted that time is represented in logarithmic scale. AGGLPlanner (in blue) is not affected by the number of symbols that the planner can create, while the time required by FastDownward (in red) grows exponentially.

aware of the existence of a coffee pot in the table. Also, to force the robot to cooperate with the human, it is assumed that the robot is unable to recognize milk containers.

Because the table was not fully inspected, the first plan the robot provides is an optimistic version in which it considers the possibility that it can find all objects in the same table:

1. Find a new object in the table.
2. Recognize the object as a mug.
3. Show the mug to the human.
4. Find another object visually.
5. Classify the new object as unknown.
6. Show the *unknown* object to the human.
7. Wait for the human to classify the object as a milk pot.

While AGGLPlanner spent 32.649 seconds in the planning process, the experiment failed to run using FastDownward or Metric-FF because of the memory consumption requirements for the instantiation phase.

Of course, when implemented in the robot the plan could fail because there could be no new objects in the table, so the execution of the plan would have to

be monitored. The goal of the experiment was not doing the experiment fast, but demonstrating that the proposed approach could be used for several purposes, including:

- Reasoning about finding and modeling new objects.
- Demonstrate how some basic features of a theory of mind can be integrated in a robot using graph-models and graph-rewriting rules *i.e.,* the robot shows the human the objects to be sure the human is aware of them. It also takes into account the different abilities of humans and robots when finding a plan.

5 CONCLUSIONS

This paper presented how graph-models and graph-rewriting rules can be used to perform HRI tasks such as the one described in section 4.2, which requires some features of a theory of mind and planning using dynamic world sizes. In order achieve this, planning systems have to take into account not only the knowledge of the robot but also its beliefs regarding the knowledge of the human it interacts with and its abilities.

Despite its simplicity AGGLPlanner achieved better performance in comparison to state-of-the-art PDDL planners. Its efficiency in comparison to the one of FastDownward is derived from the fact that AGGLPlanner (which lacks of sophisticated heuristics) does not perform grounding. It is a topic that maybe deserves more attention in the planning community.

There are two aspects in which the authors plan extending AGGL: first, including support for stochastic effects; and second, support for predicate and cost evaluation using external programs (see [19]).

References

1. Areces, C., Bustos, F., Dominguez, M., Hoffmann, J.: Optimizing planning domains by automatic action schema splitting. In: International Conference on Automated Planning and Scheduling (ICAPS2014) (2014)
2. Baron-Cohen, S.: Mindblindness: An essay on autism and theory of mind. MIT press (1997)
3. Blumenthal, S., Bruyninckx, H.: Towards a domain specific language for a scene graph based robotic world model. In: Domain-Specific Languages and Models for Robotic Systems (DSLRob), Proceedings of the Fourth International Workshop on. pp. 7–13 (2013)
4. Blumenthal, S., Bruyninckx, H., Nowak, W., Prassler, E.: A scene graph based shared 3D world model for robotic applications. In: Robotics and Automation (ICRA), 2013 IEEE International Conference on. pp. 453–460. IEEE (2013)
5. Chrpa, L.: Combining learning techniques for classical planning: Macro-operators and entanglements. 2012 IEEE 24th International Conference on Tools with Artificial Intelligence 2, 79–86 (2010)

6. Edelkamp, S.: Limits and possibilities of PDDL for model checking software. Edelkamp, & Hoffmann (Edelkamp & Hoffmann, 2003) (2003)
7. Fox, B.R..M.: Introducing a new lifted heuristics based on lifted relaxed planning graphs. In: Proceedings of the Heuristics and Search for Domain-independent Planning Workshop (ICAPS'12) (2012)
8. García, J., Florez, J.E., Torralba, A., Borrajo, D., Linares López, C., García-Olaya, A., Sáenz, J.: Combining linear programming and automated planning to solve intermodal transportation problems. European Journal of Operational Research 227, 216–226 (2013)
9. Helmert, M.: The fast downward planning system. Journal of Artificial Intelligence Research 26, 191–246 (2006)
10. Hoffmann, J.: The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables. Journal of Artificial Intelligence Research 20, 291–341 (2003)
11. Jia, Z., Gallagher, A., Saxena, A., Chen, T.: 3D-based reasoning with blocks, support, and stability. In: Conference on Computer Vision and Pattern Recognition (2013)
12. Kamps, T., Kleinz, J., Read, J.: Constraint-based spring-model algorithm for graph layout. In: Graph Drawing, Lecture Notes in Computer Science, vol. 1027, pp. 349–360. Springer (1996)
13. Korf, R.E.: Planning as search: A quantitative approach. Artificial Intelligence 33(1), 65–88 (1987)
14. Kovarsky, A., Buro, M.: A first look at build-order optimization in real-time strategy games. In: Proceedings of the GameOn Conference. pp. 18–22 (2006)
15. Lemaignan, S., Ros, R., Sisbot, E.A., Alami, R., Beetz, M.: Grounding the interaction: Anchoring situated discourse in everyday human-robot interaction. International Journal of Social Robotics pp. 1–19 (2011)
16. Manso, L.J.: Perception as Stochastic Sampling on Dynamic Graph Spaces. Ph.D. thesis, Cáceres Polytechnic School, University of Extremadura (2013)
17. McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., Wilkins, D.: PDDL: the planning domain definition language (1998)
18. Ros, R., Lemaignan, S., Sisbot, E.A., Alami, R., Steinwender, J., Hamann, K., Warneken, F.: Which one? grounding the referent based on efficient human-robot interaction. In: 19th IEEE International Symposium in Robot and Human Interactive Communication (2010)
19. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN planning for web service composition using SHOP2. Web Semantics: Science, Services and Agents on the World Wide Web 1(4), 377–396 (2004)
20. Warnier, M., Guitton, J., Lemaignan, S., Alami, R.: When the robot puts itself in your shoes. Explicit geometric management of position beliefs. In: Proceedings of the 21st IEEE International Symposium on Robot and Human Interactive Communication (2012)