Proceedings, The Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-16)

Building Helpful Virtual Agents Using Plan Recognition and Planning

Christopher Geib, Janith Weerasinghe
Sergey Matskevich, Pavan Kantharaju
Department of Computer Science
Drexel University
Philadelphia, PA 19104, USA
cgeib@drexel.edu

Bart Craenen
School of Computing Science
Newcastle University
Newcastle NE1 7RU, England, UK
bart.craenen@newcastle.ac.uk

Ronald P. A. Petrick
Department of Computer Science
Heriot-Watt University
Edinburgh EH14 4AS, Scotland, UK
r.p.petrick@hw.ac.uk

Abstract

This paper presents a new model of cooperative behavior based on the interaction of plan recognition and automated planning. Based on observations of the actions of an "initiator" agent, a "supporter" agent uses plan recognition to hypothesize the plans and goals of the initiator. The supporter agent then proposes and plans for a set of subgoals it will achieve to help the initiator. The approach is demonstrated in an open-source, virtual robot platform.

Introduction

Current non-player characters (NPC) in games and online settings are unable to meaningfully *help* a user beyond scripted interactions. To create more believable and engaging interactions, we want non-player characters that are able to act as real team-mates and partners to achieve our specific goals. Imagine, an NPC that from watching your actions can infer when it should help you by attacking the same enemy that you are attacking or provide cover for you by attacking a different enemy. In short, we want to create non-player characters that move beyond a small set of scripted helpful behaviors. This paper describes an approach to reasoning about help and an architecture that will allow non-player agents to engage in helpful activity.

However, providing help is a complex cognitive task, it requires recognizing the goals of others, recognizing when your acting can achieve open subgoals of someone else's plan, and even communicating your intentions to act in order to help. In the worst case, identifying opportunities to help, and generating an appropriate response, requires reasoning over the entire joint space of actions for all agents. Thus, the computational cost of reasoning about fully general cooperative action is impractical for all but the simplest domains.

To avoid this cost, we will restrict the actions that our method will consider. We explicitly rule out consideration of "joint actions", that is actions that require two agents to perform them like moving heavy objects. However, this is not as significant a restriction as it might initially seem. Consider two agents setting a table for dinner, where the first agent sets the plates and glasses, and the second agent sets the knives, forks, and spoons. Each agent's subgoals and actions are disjoint but together they contribute to a shared goal.

To establish terminology, this paper considers scenarios, where one agent, the *supporter*, must decide how to act to help a second agent, the *initiator*, achieve its goals. While this paper assumes the supporter is a non-player character in an interactive setting, no assumption is made about the initiator which may either be a human or synthetic agent. Further, we assume there is only one supporter agent, and we leave the question of negotiation for multiple supporters for future work.

To produce helpful behavior we propose combining *plan recognition* with *automated planning*, using a lightweight negotiation process to ensure the set of supporter goals is acceptable to both agents. Critically all three modules work with a single representation of actions and plans used in (Geib 2009; Geib and Goldman 2011). Geib's original work focuses exclusively on plan recognition. In this work, we will make use of his plan recognizer, but we will refer to it as LEX_{rec}, and introduce a planner we will call LEX_{gen} to highlight that they are based on the same action representation. As we will discuss, having a single representation removes the need to translate the representations used by different reasoning components. To the best of our knowledge, this is the first system to use the same representation for planning, plan recognition and negotiation of cooperative activities.

In this setting, the supporter will infer the plans of the initiator, specifically identifying open future subgoals that contribute to the initiator's plan. The supporter then nego-

tiates with the initiator to find a subgoal it could helpfully achieve. This involves a search in two spaces, first to confirm the initiator's goal, and second to identify an open subgoal within the inferred plan that could be performed by the supporter. Once negotiation is complete, the agreed upon goals are passed to the planner to build a sequence of actions for the supporter to execute that will help the initiator.

Notice that centralized planning for both agents is not used to enforce collaborative behavior. Neither is a "unidirectional control" relationship assumed to exist between the initiator and the supporter such that the supporter can only do what it is told to do by the initiator. For example, after observing the initiator place spoons on the table, the supporter might infer that the initiator is setting the table, and the plates still need to be set. However, if during negotiation the initiator denies the hypothesized goal (e.g., instead of setting the table, they are counting forks) or the proposed subgoal (e.g., only bowls need to be set), the supporter must find an alternative goal/subgoal pair to help the initiator.

The chief contribution of this paper is the exposition of help as a phenomena that results from the interaction of plan recognition and planning. Our approach is demonstrated in an open-source, virtual robot built using the Unity Game Engine. Thus this paper has two contributions: first, formalizing the reasoning required for producing helpful behavior, and second, a platform to explore negotiation of helpful actions by a virtual agent.

This paper is organized as follows. First, we review related work. Next, we highlight the main components in our approach: the plan recognizer, the planner, the negotiator, their shared representation and integration. We then present the results of our approach tested in three experimental domains. Finally, we discuss the limitations of our approach and highlight future directions.

Related Work

Help has been described as a primary property of a plan, or implicit in multiagent actions. For example, (Pollack 1990; Lochbaum, Grosz, and Sidner 1990) explicitly reason about coordination and helping in the form of shared plans and mutual beliefs. However, to establish such plans or beliefs requires developing shared knowledge between the agents. This is difficult to achieve in real-world settings. Another formulation of help relies on action representations for multiagent joint actions (i.e., actions that require two or more agents for their execution) (Brafman and Domshlak 2008; Boutilier and Brafman 2001). However, such representations do not address the case where helping is not a consequence of such multiagent joint actions.

There also exists significant prior work on multiagent planning (Brafman and Domshlak 2008; Brenner 2003; Crosby, Jonsson, and Rovatsos 2014). Such work understands joint action as the decentralized solving of constraint optimisation problems (Modi et al. 2003). However, this work has never directly addressed the problem of when one agent can help another agent, rather than simply working on a separate goal or plan.

Natural language dialogue as a means of coordinating actions between a robot and a human has been proposed as a partial solution to understanding help (Fong, Thorpe, and Baur 2003). The combination of natural language and goal inference has been explored for the task of selecting actions to contribute to an ongoing task, or for correcting the action of a human already engaged in the task (Foster et al. 2008; Giuliani et al. 2010). However, this work presupposes the shared knowledge of an existing task. Finally, hybrid architectures have been used to integrate diverse components with different representational requirements, particularly when robotic agents must cooperate with a human (Hawes et al. 2007; Kennedy et al. 2007; Zender et al. 2007). Here we propose a shared representation for all the components eliminating the problem.

Approach

As we have suggested, our approach to understanding helpful behavior relies on plan recognition, planning, and negotiation using a shared action representation. Next we describe the shared action representation and each of these components.

Representing Actions and Plans

This work builds on the work of (Geib 2009) on plan recognition and the idea that an *action* can be thought of as a function from states of the world to states of the world. Taking this idea seriously, we will use a functional grammar formalism called Combinatory Categorial Grammars (CCGs) (Steedman 2000) to define the actions and possible plans in a domain.

First, consider a simple action, $\mathbf{a}_i$, that an agent can execute. Suppose that, regardless of the initial state of the world, when $\mathbf{a}_i$ is executed, it causes a transition to a state s_A. Thus $\mathbf{a}_i$ is a function that, regardless of the initial state, results in a transition to s_A. We can define an *atomic CCG category* A, as a zero arity function that results in a transition from an state to s_A and we could define the action as having this functional category

$$\mathbf{a}_i := A.$$

However, very few actions result in a known transition in every context, therefore we introduce *complex CCG categories*. Imagine a second action, $\mathbf{a}_j$, such that from state, s_B, results in a transition to s_A. We can capture such a function in a *complex category* defined using the backslash operator

$$\mathbf{a}_j := A \backslash \{B\}.$$

This defines $\mathbf{a}_j$ as function that has a single *argument*, B, that must occur temporally before it in order to produce the *result*, A. We could also imagine a third action, $\mathbf{a}_k$, that results in transition to state s_A but only if an action taken after it achieves s_B (e.g. consider unlocking a door resulting in the door being open, but only if the door is pulled open after it is unlocked.) We can capture this action's complex category with the forward slash operator

$$\mathbf{a}_k := A / \{B\}.$$

The direction of the slash operators indicates where the category's argument must occur (temporally before or after) for the function to produce its result.

$$\textbf{grasp} := (Hold/\{Lift\})\backslash\{Reach\}$$

Figure 1: A category definition for the action **grasp**

$$\begin{aligned}
\textbf{set-forks} &:= SetTable/\{SetKnvs, SetSpns, SetPlts, SetGlsss\} \mid \\
&\quad (CleanFrks/\{PutAwayFrks\})/\{WashFrks\}. \\
\textbf{set-knives} &:= SetKnvs. \quad \textbf{set-spoons} := SetSpns. \\
\textbf{set-plates} &:= SetPlts. \quad \textbf{store-forks} := PutAwayFrks. \\
&\vdots \\
\textbf{reach} &:= Reach. \quad \textbf{lift} := Lift. \\
\textbf{grasp} &:= (Hold/\{Lift\})\backslash\{Reach\}
\end{aligned}$$

Figure 2: Portion of a CCG action Lexicon.

Note that since the slash operators are defined over categories, we can now recursively define actions as functions of arbitrary arity that have arguments in both directions. For example we could define the action **grasp** as having a functional category that results in *Hold* but only if before it *Reach* is achieved and after it *Lift* is achieved. See Figure 1. Effectively we have defined a Curried function(Curry 1977) of arity two that has an argument both before and after it. Note, in the implemented system both actions and categories can have typed parameters that allow them to specify objects they act on. However for simplicity of exposition, we will use a propositional representation.

Following (Geib 2009), we will call a set of associations of actions to categories an *action lexicon.* Further, following (Baldridge 2002) we allow arguments within a set of braces to be unordered with respect to each other. Thus in Figure 2, actions with the categories *SetKnvs*, *SetSpns*, *SetPlts*, and *SetGlsss* all need to occur after the action **set-forks** to achieve *SetTable*, but are unordered with respect to each other. Note that in this paper, our running example of helpful behavior will be based on setting the table. Thus Figure 2 captures a portion of the lexicon for this domain. Further we note that the action **set-forks** has more than one possible category assigned to it in the lexicon. One in which the result category is *SetTable* and one in which it is *CleanFrks*.

Plan Recognition with CCGs

We distinguish work in *activity recognition* (also called *goal recognition*) (Liao, Fox, and Kautz 2005; Hoogs and Perera 2008; Blaylock and Allen 2003) and *plan recognition*. Activity recognition creates a single unstructured label representing the overarching goal of the observed actions. For example, activity recognition would label a partial sequence of pick and place actions of forks, knives, spoons, and plates as an instance of setting the table. This kind of single label is insufficient for our purposes. To reason about help, we must know which steps in the plan are already completed, and which steps the supporter can still contribute to.

Plan recognition identifies not only the goal being pursued by the agent but also the subgoals of the plan that have already been accomplished, and those predicted to be future steps in the plan. Thus, a plan recognition algorithm produces the complete unexecuted frontier of a hierarchical plan for the goal(Kautz 1991; Blaylock and Allen 2003; Geib 2009). For example, after observing picking and placing forks followed by knives, a plan recognition engine would identify that the goal was to set the table, the current subgoal was to set the knives, and that in the future, the agent would be setting spoons, plates, and glasses.

(Geib 2009) views the plan recognition problem as an instance of parsing observed actions based on a probabilistic CCG lexicon that specifies the set of plans to be recognized. Parsing is performed by using rightward and leftward function application and rightward function combination over pairs of categories until only a single result remains or no other operations area possible. For example, consider the derivation in Figure 3 that parses the observation sequence: [**reach**, **grasp**, **lift**] using the lexicon in Figure 2.

$$\dfrac{\dfrac{\textbf{reach}\quad\textbf{grasp}}{Reach \quad ((Hold)/\{Lift\})\backslash\{Reach\}}< \quad \textbf{lift}\quad Lift}{\dfrac{(Hold)/\{Lift\}}{Hold}>}$$

Figure 3: Parsing Observations with CCG categories using function application

As each observation is encountered, it is assigned one of its categories from the lexicon. Rightward and leftward application are then used in this case to combine the categories. Parsing can then be seen as a search over the possible assignments of categories to actions, function applications, and compositions. More details of this approach can be found in (Geib 2009; Geib and Goldman 2011).

Given a set of observed actions, and a CCG action lexicon, LEX_{rec} performs probabilistic plan recognition using this parsing algorithm. It does this by weighted model counting. Given a set of observations, it generates the complete set of possible parses for the observed actions consistent with the grammar, along with a probability for each. These category structures represent the hypothesized plans being executed by the agent. From them we can extract an ordered set of subgoals that must still be executed for the result category to be achieved, associated with the probability of the hypothesis.

Note that LEX_{rec} supports the possibility that a given agent can be pursuing multiple, partially ordered plans. Therefore, we will represent each hypothesized parse as a pair:

$$(P, [C_i^+]),$$

where P is the probability of the hypothesis, and the C_i represents the categories capturing any hypothesized plans. For example, three possible hypotheses from the table setting example (after observing the setting of forks) could be:

$(.95, [\{SetTable/\{SetKnvs, SetSpns, SetPlts, SetGlsss\}\}]),$
$(.045, [\{(CleanFrks/\{PutAwayFrks\})/\{WashFrks\}\}]),$
$(.005, [\{CountingFrks\}]).$

The first pair captures the hypothesis that with 95% probability the agent is following a plan to set the table, and still has the subgoals to set the knives, spoons, and plates. The second pair captures the hypothesis that with 4.5% probability the agent is cleaning the forks and still needs to wash them and put them away (in that order). The third pair captures the hypothesis that with only 0.5% probability the agent is simply counting the forks and is done with its plan. Thus, each hypothesis provides us with access to the probability of the plans being executed, the goals they are intended to achieve, and the subgoals in the plan that have yet to be achieved. This is the information that we need in order to identify where the supporter can help the initiator. We will discuss this next.

Subgoal Identification and Negotiation

Negotiating collaboration is a two step process. First a supporter must confirm the objective of the initiator's high-level plan. Without this, the supporter could waste significant amounts of time suggesting subgoals it could achieve, but do not contribute to the initiator's goal and plan. Using the hypothesis structures from LEX_{rec} this is straightforward.

When a single plan is being pursued by the initiator, sorting the hypotheses by their probabilities ranks the goals of the plans being pursued by their likelihood. The negotiator can extract the result of the complex category in the hypothesis and query the initiator to verify if it is their actual goal. Having identified the initiator's goal, the supporter can then attempt to identify an open future subgoal within the hypothesized plan. For example, in the fork setting scenario, the most likely hypothesis is

$$(.95, [\{\textit{SetTable}/\{\textit{SetKnvs}, \textit{SetSpns}, \textit{SetPlts}, \textit{SetGlsss}\}\}]).$$

After confirming that *SetTable* is the initiator's goal, the supporter can suggest that it take on the subgoals of *SetKnvs*, *SetSpns*, *SetPlts*, and *SetGlsss*, or a subset thereof. A maximally helpful agent will volunteer to do all of these subgoals. However, the negotiation process could result in a number of other outcomes, in which the supporter performs some subset of the subgoals, or none of them at all.

It is worth noting, in this process the actual plan inferred by the supporter is never shared with the initiator. This means that there can be significant differences between the initiator's beliefs about the role of the subgoal the supporter is going to execute in the joint plan and the supporter's beliefs about this. However, as long as both agents have agreed on the initiator's goal and the subgoal to be performed by the supporter we can say that the supporter is at least attempting to help the initiator achieve their goal.

The process of negotiating collaboration is then a directed search. First we search to identify the goal of the initiator's plan, and then to find appropriate subgoals from the set of known unaccomplished subgoals of the plan the supporter has inferred for the goal. Conducting this search is the role of the negotiator component in our system.

1 *Hold*

2 $((\textit{Hold}/\{\textit{Lift}\})\backslash\{\textit{Reach}\}) : \textbf{grasp}$

3 $\textit{Reach}$ $((\textit{Hold}/\{\textit{Lift}\})\backslash\{\textit{Reach}\}) : \textbf{grasp}$ $-1<$

4 **reach**

5 $(\textit{Hold}/\{\textit{Lift}\}) : \textbf{reach}, \textbf{grasp}$ *append-L*

6 $\textit{Hold} : \textbf{reach}, \textbf{grasp}$ $\textit{Lift}$ $-1<$

7 **lift**

8 $\textit{Hold} : \textbf{reach}, \textbf{grasp}, \textbf{lift}$ *append-R*

Figure 4: Building a plan to achieve the category *Hold*. As each argument is removed from the category an action that achieves it is added to the front or the back of the plan.

Automated Planning with CCGs

Once negotiation is complete and has produced a set of subgoals for helping the initiator, the supporter must generate a concrete sequence of actions to execute in the world. To do so, we use a new planner, LEX_{gen}(Geib 2016), that uses a CCG action lexicon to guide the search for a plan.

Like other planners, a LEX_{gen} planning problem consists of an initial state, a goal state, and a set of actions. The initial state is simply the planner's model of the initial state of the world. Goals are a statement of a set of conditions that the planner is trying to achieve. LEX_{gen} also has a set of precondition-postcondition *projection rules* for each action that define how the world changes if the action is executed. However, unlike other planners, LEX_{gen} also has a CCG action lexicon.

As we have already seen, each entry in an action lexicon defines a direct link from a basic executable action to a state that can be achieved by the action in the form of atomic or complex categories. The main insight behind LEX_{gen} is that the structure of the categories assigned to an action in the lexicon provides information about how to build a plan to achieve the state captured by the result of each category assigned to each action.

Consider a case where we want to build a plan to achieve *Hold*. The definition found in Figure 1 tells us that we can build a plan to achieve this by executing **grasp**, as long as before it we achieve a *Reach* and after *Lift*. LEX_{gen} uses this information to build plans by following the structure of the categories. For example, Figure 4 shows LEX_{gen} building the complete plan for *Hold*. Note that unlike other planners, actions can be added both at the front and the end of the current plan.

Having covered how an individual plan is built, LEX_{gen} works in our system by being passed a category to achieve by the negotiator. LEX_{gen}then searches over the space of all of the realizations of all of the categories that have the passed category as its result. Once a complete plan is built, the action projection rules are used to verify that the plan achieves the desired goal. If it does, the search for a plan stops. If not, the search backtracks to consider alternatives.

Effectively, the lexicon provides heuristic guidance for the search for a plan. A complete discussion of LEX_{gen} is beyond the scope of this paper. We refer the interested reader to (Geib 2016) for more details, but for our use, it is critical

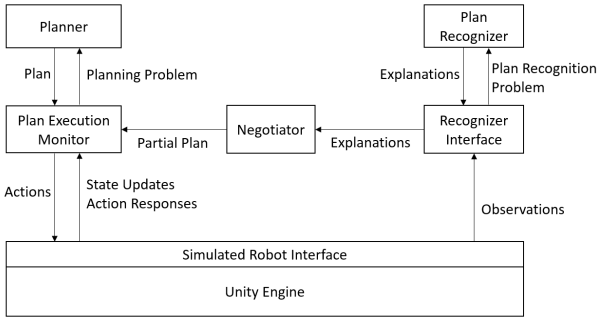

Figure 5: Information flow diagram for our framework.

that the same action lexicon used for plan recognition is used to guide planning. This means that no translation of terms or even format is required for the negotiator to pass categories from the recognizer to the planner. This greatly simplifies the system's construction.

Integration and Operation

LEX_{rec}, LEX_{gen}, and the negotiator are implemented as C++ libraries that expose APIs through ZeroC's Internet Communication Engine (ICE), a modern distributed computing platform (Henning 2004). This modularity allows each of the components to be used as standalone services by a client application implementing the framework in a traditional client-server architecture. It also will allow us to conduct future studies where these components could be replaced by a different implementations.

Figure 5 illustrates the flow of control between the plan recognition, negotiation, and automated planning components in the framework. At system initialization, both the plan recognizer and planner are provided with the same domain-dependent knowledge in the form of the action lexicon. The system is then provided with a set of observations of the initiator's actions. These are fed into LEX_{rec}, which produces a set of hypotheses structures that contain the initiator's high-level goal and plan. This structure is then handed over to the negotiation process which mediates the negotiation between the supporter and initiator. Negotiation proceeds by applying directed search to the hypothesis structure to produce a set of goals for the planner. The agreed upon goals for the supporter are then passed to LEX_{gen} to generate plans that are then executed by the virtual robot implemented in the unity game engine and shown in Figure 6. Again notice that all three of the reasoning components are using the same representation of plans and actions, and therefore no translation is required between these components.

Experimental Demonstration and Validation

We now present three scenarios based on the table-setting domain, to demonstrate our framework. The initial state and observations for the three scenarios remains the same: the initiator picks up two forks and and puts them down in their appropriate positions on the table. The scenarios differ in the

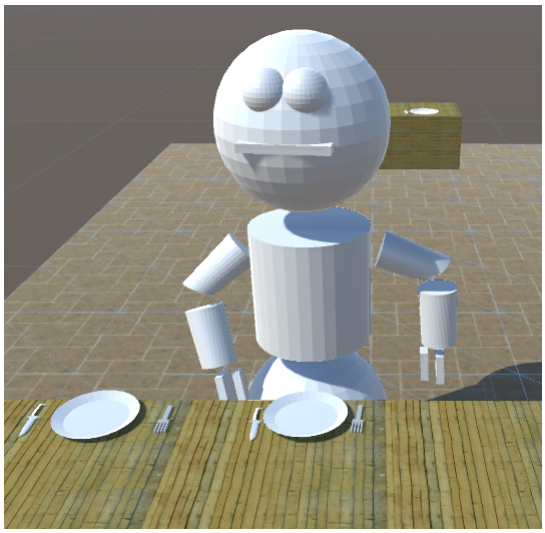

Figure 6: Virtual robot implemented in Unity game engine in a simple work environment.

results of the negotiation process. The process ends when the planner builds a plan for the supporter to perform, and the virtual robot executes the actions.

These scenarios are validated at two points: first, whether LEX_{rec} correctly interprets the observations, and second, whether LEX_{gen} produces the correct plans. The example scenarios are designed so that we know how the negotiation process will go and the correct results of the planning process.

Note that total processing time required for each of these, admittedly small-scale, scenarios, on contemporary hardware, is below a second. For larger scenarios, and more ambiguous domains, this is expected to increase. Prior work with LEX_{rec}(Geib 2009) shows that it scales very well even with large domains. Our initial experience with LEX_{gen} indicates that it has state of the art runtimes and scaling. Finally, the negotiator's runtime is bounded by the interaction time with the user. All of this gives us good reason to believe that our system will scale to significantly larger problem domains.

Scenario 1: The plan recognizer correctly identifies the initiator's goal, and the initiator wants all of the outstanding subgoals done by the supporter. The first hypothesis the negotiator considers is:

$$(0.95, [\{\mathit{SetTable}/\{\mathit{SetKnvs}, \mathit{SetSpns}, \mathit{SetPlts}, \mathit{SetGlsss}\}\}]).$$

In this case, there is no need for any significant search by the negotiator. The negotiation takes the following form:

1. Supporter: Are you setting the table?
2. Initiator: Yes.
3. Supporter: Do you want me to set the knives?
4. Initiator: Yes.
5. Supporter: Do you want me to set the spoons?
6. Initiator: Yes.
7. Supporter: Do you want me to set the plates?
8. Initiator: Yes.

9. Supporter: Do you want me to set the glasses?
10. Initiator: Yes.
11. Supporter: I will now help you set the table.

Once completed, the *SetKnvs*, *SetSpns*, *SetPlts*, and *SetGlsss* categories are passed, without modification, into LEX_{gen} and the planner attempts to generate plans to achieve the given categories. Since the hypothesis and the resulting plan are both known beforehand, we can verify that the system results match the desired outcome.

Scenario 2: Tests directed search to correctly identify the initiator's goal from hypotheses provided by LEX_{rec}. In this case, the search focuses on high-level goal identification, with the initiator rejecting the hypothesis initially presented by the supporter.

As in the last scenario, the most likely hypothesis is:

$$(0.95, [\{\textit{SetTable}/\{\textit{SetKnvs}, \textit{SetSpns}, \textit{SetPlts}, \textit{SetGlsss}\}\}]).$$

However, when queried about the goal of *SetTable*, the initiator rejects this goal. The negotiator therefore considers less likely hypothesis until the correct one is found. In this case, considering the next most likely hypothesis:

$$(0.045, [(\{\textit{CleanFrks}/\{\textit{WashFrks}\})/\{\textit{PutAwayFrks}\}\}]),$$

correctly identifies the initiator's goal to be that of cleaning the forks, and negotiation proceeds as shown below:

1. Supporter: Are you setting the table?
2. Initiator: No.
3. Supporter: Are you cleaning the forks?
4. Initiator: Yes
5. Supporter: Do you want me to wash the forks?
6. Initiator: Yes.
7. Supporter: Do you want me to put the forks away?
8. Initiator: Yes.
9. Supporter: I will now help you clean the forks.

The subgoals are passed to LEX_{gen}, the planner builds plans for cleaning and putting away the forks, and the virtual robot performs the plan.

Scenario 3: The initiator's goal and plan is correctly identified, but one (or more) of the subgoals is rejected by the initiator. In this case, the supporter, using directed search of the hypothesis structure, negotiates with the initiator until it finds subgoals it can perform or finds there are none. As in the first scenario the initial hypothesis is:

$$(0.95, [\{\textit{SetTable}/\{\textit{SetKnvs}, \textit{SetSpns}, \textit{SetPlts}, \textit{SetGlsss}\}\}]).$$

and negotiation now takes the following form:

1. Supporter: Are you setting the table?
2. Initiator: Yes.
3. Supporter: Do you want me to set the knives?
4. Initiator: Yes.
5. Supporter: Do you want me to set the spoons?
6. Initiator: No, I will do that myself.
7. Supporter: Do you want me to set the plates?
8. Initiator: No, I will do that myself.
9. Supporter: Do you want me to set the glasses?
10. Initiator: Yes.
11. Supporter: I will now help you set the table.

As is expected, only the goals for the knives and glasses are passed to LEX_{gen} for planning and execution. This demonstrates that the initiator is not limited to letting the supporter perform all the remaining subgoals in a hypothesis. Our framework provides enough flexibility for the initiator to decide how, they want to be helped, without the need for elaborate reasoning on the part of the supporter.

Discussion

The three experimental scenarios demonstrate that our approach successfully generates cooperative plans. For each scenario, LEX_{rec} interprets the observations correctly, supplying the correct hypothesis structure to the negotiation process; and the negotiator subsequently presents $\text{LEX}_{\text{gen}}$with the needed subgoals, with the planner producing the correct plans for execution on our virtual robot.

However, the framework also relied on certain assumptions concerning the knowledge of the initiator and supporter. Note, the full structured plan that is inferred by the supporter is never shared with the initiator. It is assumed that if the goal of the plan is agreed on and the subgoal is agreed on then the correspondence between the initiator's version of the plan and the supporters version of the plan are close enough to each other to make the actions the supporter does helpful to the initiator's goal.

The frame work also assumes referential correspondence. It is possible that different agents might use different terms to refer to the same objects, actions, or goals. If there is sufficient disagreement on such terms, negotiation will simply break down in the face of failed communication. Likewise, a high degree of overlap between the knowledge of the agents, and a tighter correspondence in the names used to identify domain concepts, should give rise to situations where cooperation is more easily negotiated. This approach assumes close correspondence between the initiator's and the supporter's terminology used in negotiation.

This said, it is worth noting a final time that because all three of the major reasoning components in our system are using the same representation for actions there is no referential problem or need for translating between these components. This significantly simplified our system building, and will greatly reduce the effort to move the system from one domain to another.

It is worth noting that this approach does not generate plans with joint actions, that is actions where multiple agents must coordinate to perform the same task (e.g., lifting a table). Instead, it only generates independent action sequences for the supporter once there is mutual agreement as to the supporter's subgoals. This is an area for future work. There is also future work to be done in this framework on scheduling of coordination. For example, if in the fork cleaning example the initiator says they will wash the forks but allows the supporter to put them away, it is important that the supporter wait until the initiator has finished their task before beginning.

In this work, we have focused on the importance of the supporter being *proactive* in suggesting subgoals that could

help the initiator. We specifically avoided having the supporter ask the initiator how it can help. If the initiator is a human, and the supporter is an artificial agent, such an approach can force the human to respond to a large number of requests (including clarifications) both as to what they are doing and how the supporter could be of assistance. That said, in cases where the initiator chooses to tell the supporter what it wants done, this information can be passed directly from the negotiator to the planner.

Conclusion

This paper presented a framework for combining plan recognition and automated planning using the same action representation to produce cooperative behaviour between a pair of agents. Successful integration of the plan recognition and planning components centered around appropriate subgoal identification by the plan recognizer, combined with a lightweight negotiation process which generated goals to be used by the planner for constructing appropriate action sequences. The generated plans were then demonstrated by execution on a virtual robot platform. A set of experiments demonstrated the potential of our approach, and helped motivate our ongoing and future work to extend these techniques to more complex real-world situations.

Acknowledgements

The research leading to these results has received funding from the European Union's Seventh Framework Programme under grant no. 270273 (XPERIENCE, xperience.org) and grant no. 610917 (STAMINA, stamina-robot.eu).

References

Baldridge, J. 2002. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. Ph.D. Dissertation, University of Edinburgh.

Blaylock, N., and Allen, J. 2003. Corpus-based statistical goal recognition. In *Proc. of IJCAI 2003*, 1303–1308.

Boutilier, C., and Brafman, R. 2001. Partial-order planning with concurrent interacting actions. *JAIR* 14:105–136.

Brafman, R., and Domshlak, C. 2008. From one to many: Planning for loosely coupled multi-agent systems. In *Proc. of ICAPS 2008*, 28–35.

Brenner, M. 2003. A Multiagent Planning Language. In *Proceedings of the Workshop on PDDL at ICAPS 2003*.

Crosby, M.; Jonsson, A.; and Rovatsos, M. 2014. A single-agent approach to multiagent planning. In *Proc. of ECAI 2014*, 237–242.

Curry, H. 1977. *Foundations of Mathematical Logic*. Dover Publications Inc.

Fong, T.; Thorpe, C.; and Baur, C. 2003. Collaboration, dialogue, and human-robot interaction. In *Robotics Research, Volume 6 of Springer Tracts in Advanced Robotics*. Springer. 255–266.

Foster, M. E.; Giuliani, M.; Müller, T.; Rickert, M.; Knoll, A.; Erlhagen, W.; Bicho, E.; Hipólito, N.; and Louro, L. 2008. Combining goal inference and natural-language dialogue for human-robot joint action. In *ECAI Workshop on Combinations of Intelligent Methods and Applications*.

Geib, C., and Goldman, R. 2011. Recognizing plans with loops represented in a lexicalized grammar. In *Proceedings of the twenty fifth Conference on Artificial Intelligence*, 958–963. San Francisco, CA, USA: AAAI Press.

Geib, C. W. 2009. Delaying commitment in probabilistic plan recognition using combinatory categorial grammars. In *Proc. of IJCAI 2009*, 1702–1707.

Geib, C. W. 2016. Lexicalized reasoning about actions. *Advances in Cognitive Systems* Volume 4:187–206.

Giuliani, M.; Foster, M. E.; Isard, A.; Matheson, C.; Oberlander, J.; and Knoll, A. 2010. Situated reference in a hybrid human-robot interaction system. In *Proc. of INLG*, 67–75.

Hawes, N.; Sloman, A.; Wyatt, J.; Zillich, M.; Jacobsson, H.; Kruijff, G.-J. M.; Brenner, M.; Berginc, G.; and Skočaj, D. 2007. Towards an integrated robot with multiple cognitive functions. In *Proc. of AAAI 2007*, 1548–1553.

Henning, M. 2004. A new approach to object-oriented middleware. *IEEE Internet Computing* 8(1):66–75.

Hoogs, A., and Perera, A. A. 2008. Video activity recognition in the real world. In *Proc. of AAAI 2008*, 1551–1554.

Kautz, H. A. 1991. A formal theory of plan recognition and its implementation. In Allen, J. F.; Kautz, H. A.; Pelavin, R. N.; and Tenenberg, J. D., eds., *Reasoning About Plans*. Morgan Kaufmann. 69–126.

Kennedy, W. G.; Bugajska, M. D.; Marge, M.; Adams, W.; Fransen, B. R.; Perzanowski, D.; Schultz, A. C.; and Trafton, J. G. 2007. Spatial representation and reasoning for human-robot collaboration. In *Proc. of AAAI 2007*, 1554–1559.

Liao, L.; Fox, D.; and Kautz, H. A. 2005. Location-based activity recognition using relational Markov networks. In *Proc. of IJCAI 2005*, 773–778.

Lochbaum, K.; Grosz, B.; and Sidner, C. 1990. Models of plans to support communication: An initial report. In *Proc. of AAAI 1990*, 485–490.

Modi, P. J.; Shen, W.-M.; Tambe, M.; and Yokoo, M. 2003. An asynchronous complete method for distributed constraint optimization. In *Proc. of AAMAS 2003*, 161–176.

Pollack, M. 1990. Plans as complex mental attitudes. In Cohen, P.; Morgan, J.; and Pollack, M., eds., *Intentions in Communication*. MIT Press. 77–103.

Steedman, M. 2000. *The Syntactic Process*. MIT Press.

Zender, H.; Jensfelt, P.; Óscar Martínez Mozos; Kruijff, G.-J. M.; and Burgard, W. 2007. An integrated robotic system for spatial understanding and situated interaction in indoor environments. In *Proc. of AAAI 2007*, 1584–1589.