MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Using Plan Recognition in Human-Computer Collaboration

Neal Lesh, Charles Rich, Candace L. Sidner

TR98-23 December 1998

Abstract

Human-computer collaboration provides a practical and useful application for plan recognition techniques. We describe a plan recognition algorithm which is tractable by virtue of exploiting properties of the collaborative setting, namely: the focus of attention, the use of partially elaborated hierarchical plans, and the possibility of asking for clarification. We demonstrate how the addition of our plan recognition algorithm to an implemented collaborative system reduces the amount of communication required from the user.

Proceedings of Seventh Int. Conf. on User Modeling, Banff, Canada, June 1999.

MERL – A MITSUBISHI ELECTRIC RESEARCH LABORATORY
http://www.merl.com

Using Plan Recognition in Human-Computer Collaboration

Neal Lesh   Charles Rich   Candace L. Sidner*

TR-98-23   December 1998

Abstract

Human-computer collaboration provides a practical and useful application for plan recognition techniques. We describe a plan recognition algorithm which is tractable by virtue of exploiting properties of the collaborative setting, namely: the focus of attention, the use of partially elaborated hierarchical plans, and the possibility of asking for clarification. We demonstrate how the addition of our plan recognition algorithm to an implemented collaborative system reduces the amount of communication required from the user.

Submitted to Seventh Int. Conf. on User Modeling, Banff, Canada, June 1999.

*Lotus Development Corporation

Publication History:–

1. First printing, TR-98-23, December 1998

Using Plan Recognition in Human-Computer Collaboration

Neal Lesh,* Charles Rich,* Candace L. Sidner†

* MERL–A Mitsubishi Electric Research Laboratory (lesh@merl.com)

† Lotus Development Corporation

Abstract

Human-computer collaboration provides a practical and useful application for plan recognition techniques. We describe a plan recognition algorithm which is tractable by virtue of exploiting properties of the collaborative setting, namely: the focus of attention, the use of partially elaborated hierarchical plans, and the possibility of asking for clarification. We demonstrate how the addition of our plan recognition algorithm to an implemented collaborative system reduces the amount of communication required from the user.

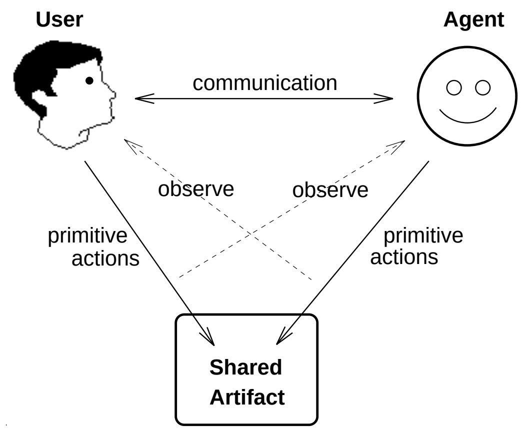
1 Introduction

An important trend in recent work on human-computer interaction and user modeling has been to view human-computer interaction as a kind of *collaboration* (e.g, [5, 10, 19, 20]). In this approach, the human user and the computer (often personified as an "agent") coordinate their actions toward achieving shared goals. A common setting for collaboration, illustrated in Figure 1(a), and which is the focus of this paper, is when two participants can both communicate with each other and observe each other's actions on some shared artifact.

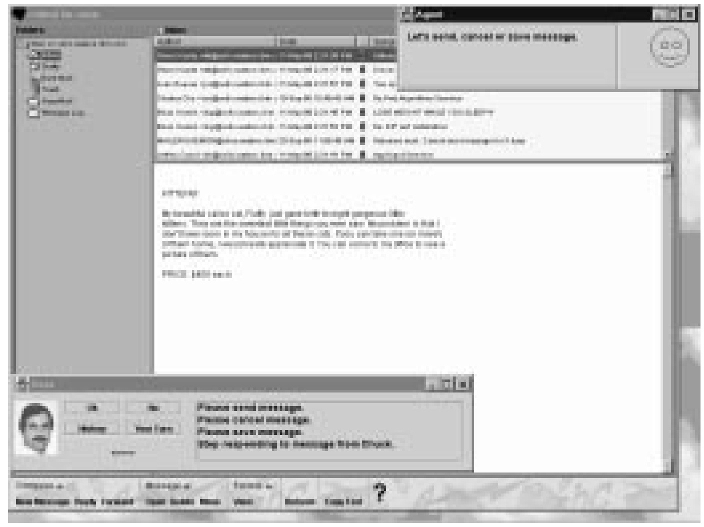
Successful collaboration requires the participants to have substantial mutual understanding of their shared goals and the actions intended to achieve them (these are part of what Grosz and Sidner [9] call the *SharedPlan*). One way to maintain this mutual understanding is through verbal communication—the participants can explicitly discuss each goal and the method they propose to achieve it. However, it is often more efficient and natural to convey intentions by performing actions. For example, if two people are attempting to get into their locked car and one picks up a brick, the other can infer that the proposed plan is for the first person to smash a window, reach in, and unlock the door. *Plan recognition* (e.g., [3, 11]) is the term generally given to the process of inferring intentions from actions.

Although plan recognition is a well-known feature of human collaboration, it has proven difficult to incorporate into practical human-computer collaboration systems due to its inherent intractability in the general case (see Section 2.1). In this work, we describe how to exploit properties of the collaborative setting in order to make plan recognition practical. Specifically, the properties we exploit are: the focus of attention, the use of partially elaborated hierarchical plans, and the possibility of asking for clarification.

We demonstrate our approach in the context of an implemented collaborative system for email. Section 1.1 below presents two example sessions with this system, which illustrate both the general idea of human-computer collaboration and how plan recognition reduces the user's communication burden. Section 2 describes the underlying plan recognition and discourse interpretation algorithms in detail. Section 3 presents results from preliminary experiments in the email domain. We then discuss related work and conclude.



(a) A setting for human-computer collaboration.



(b) Graphical interface for Collagen email collaboration.

	without plan recognition	*with plan recognition*
1	USER: "Let's work on email."	USER: "Let's work on email."
2	AGENT: "You have 5 unread messages."	AGENT: "You have 5 unread messages."
3	USER: "Let's read a message."	
4	USER: <Select message from Chuck>	USER: <Select message from Chuck>
5	USER: "Let's read message in its own window."	
6	USER: <Open message>	USER: <Open message>
7	USER: "Let's do something using message."	
8	USER: <Visit http://foo.bar.com>	USER: <Visit http://foo.bar.com>
9	...long lunch...	...long lunch...
10	USER: "What was I doing?"	USER: "What was I doing?"
11	AGENT: "Working on email.	AGENT: "Working on email.
12	Reading message from Chuck.	Reading message from Chuck.
13	Reading message in its own window.	Reading message in its own window.
14	Done something using message.	Done something using message.
15	<Visit http://foo.bar.com>	<Visit http://foo.bar.com>
16	Expecting to react to message (opt)."	Expecting to react to message (opt)."
17	USER: "Let's react to message."	
18	USER: "Let's send email."	
19	USER: <Start new message>	USER: <Start new message>
20	USER: "Let's fill in message."	
21	USER: <Address message to Dan>	USER: <Address message to Dan>
22		AGENT: "Why are you sending email to Dan?
23		(a) reacting to message from Chuck
24		(b) as a new goal "
25		USER: "I am reacting to message from Chuck."
26	...additional editing actions...	...additional editing actions...
27	USER: "Done filling in message."	USER: "Done filling in message."
28	AGENT: "Let's send, cancel or save message."	AGENT: "Let's send, cancel or save message."
29	USER: "Please save message."	USER: "Please save message."
30	AGENT: <Save message>	AGENT: <Save message>

Key: "utterance" <primitive action>

(c) Collagen email session with and without plan recognition.

Figure 1.

1.1 Collagen Email Example

Collagen [19] is an application-independent collaboration manager based on the SharedPlan theory of task-oriented collaborative discourse [16, 17]. We are currently experimenting with Collagen in several different application areas, including air travel (see [8, 19]) and email.

Figure 1(b) shows how the abstract setting for human-computer collaboration in Figure 1(a) is instantiated using Collagen in the email domain. The large window in Figure 1(b) is the graphical interface to the email part of Lotus eSuite^{tm}; this is the "shared artifact" of the collaboration. The two smaller, overlapping windows in the corners of Figure 1(b) are the agent's and user's *home windows*, through which they communicate with each other.

For an application-independent tool like Collagen, a key step in building a collaborative agent is to develop a detailed task model for the domain. Based on empirical study of people working on email, Sidner and colleagues have formalized the task structure of this domain in terms of high-level goals, such as "working on email", lower-level goals, such as "filling in a message," and primitive actions corresponding to individual clicks on the eSuite interface.

Without Plan Recognition. Let us look first at the left column of Figure 1(c), which shows how Collagen functions without plan recognition. In the first part of this session (lines 1–8) the user has the initiative. Notice how the user laboriously announces each goal before performing a primitive action which contributes to achieving it. Without plan recognition, this is the only way to maintain the mutual understanding necessary for successful collaboration.

A simple example of collaboration occurs after the user returns from a long lunch (line 9). At this point, the user's earlier intentional state is not immediately evident from the state of the graphical interface, which would show only a browser window (resulting from clicking on a URL in the message being read) with the email window behind or below it. Based on the user's earlier announcement of goals, however, the agent has constructed a SharedPlan, which it can communicate back to the user (lines 11–16) to help him reorient to what he was doing and what is expected next.[1]

The user now continues as before, announcing his subgoals as he goes, until line 27, when he declares that he is done with the goal of filling in the new message started in line 19. The agent uses this as an opportunity to suggest some expected actions (line 28), one of which the user requests the agent to perform (lines 29–30).

With Plan Recognition. The right column of Figure 1(c) shows the same task as the left column, but with our plan recognition algorithm incorporated into Collagen. Notice that, unlike the previous example, the user in this session is not required to announce each goal and subgoal before performing a primitive act (i.e., lines 3, 5, 7, 17, 18, and 20 are missing). Nevertheless, as we see in lines 11–16, the agent constructs the same SharedPlan as before.

Plan recognition does not, however, totally eliminate the need for communication about intentions. In particular, collaborators must ask for clarification when there is ambiguity regarding how to interpret some given actions. For example, the user's actions in lines 19 and 21 are consistent with two possible intentions: by sending a message to Dan, the user may either be reacting to the message from Chuck (for example, if Chuck suggested sending email to Dan) or be starting a new, unrelated email goal. The agent interrupts the user at

[1] The agent's communication in lines 11–16 derives from a more general capability in Collagen for maintaining what is called a *segmented interaction history*—see [19].

line 22 to resolve this ambiguity.[2] Section 2.2 discusses strategies for composing clarification questions.

1.2 The Role of Plan Recognition in Collaboration

This section previews the main points presented in the remainder of the paper, abstracted away from the details of the example above.

According to SharedPlan theory, a key component of the mental state of each participant in a collaboration is a set of beliefs about the mutually believed goals and actions to be performed, and about the mutually believed capabilities, intentions, and commitments of each participant. Each participant updates this set of beliefs, called the *SharedPlan*, based in part on communication with and observation of the other participants. Each participant also knows a set of methods, called *recipes*, for decomposing goals into subgoals.

Generally speaking, the role of plan recognition in this framework is as follows: Suppose one participant, e.g., the software agent, observes another participant, e.g., the user, perform an action A. The agent invokes plan recognition to determine the set of possible extensions to its current SharedPlan which are consistent with its recipe knowledge and include the user performing A. If there is exactly one possible such extension, the agent adopts this extension as its new SharedPlan; otherwise, it may ask a clarification question. A similar story can be told if the user does not actually perform A, but only proposes doing A (as in, "Let's do A").

We exploit three properties of the collaborative setting to make this use of plan recognition tractable. The first property is the *focus of attention*. When the user says, "Let's work on email," he is not only proposing a certain action be performed, he is also establishing a new context which restricts the interpretation of future utterances and actions. The full implications of focus of attention are beyond the scope of this paper (see [9]); in this work we use the weaker notion of the "focus act[3]" to limit the search required for plan recognition.

A second property of collaboration we exploit is that the processes of developing, communicating about, and executing plans are interleaved. Consequently, both the input and output of the plan recognizer are partially elaborated hierarchical plans. Unlike the "classical" definition of plan recognition (e.g., [11]), which requires reasoning over complete and correct plans, our recognizer is only required to incrementally extend a given plan.

Third, it is quite natural during collaboration to ask for clarification, either because of inherent ambiguity, or simply because the computation required to understand an observed or mentioned action is beyond a participant's abilities. We use clarification to ensure that the number of actions the plan recognizer must interpret will always be small.

2 Algorithms

This section presents our plan recognizer and describes how it is used in discourse interpretation. We begin by adopting a straightforward formalization of actions, plans, and recipes.

Let $\mathcal{ACT}$ be a set of actions which includes primitive actions, $\mathcal{PRIM} \subseteq \mathcal{ACT}$, and "top level" actions, $\mathcal{TOP} \subseteq \mathcal{ACT}$, which might be done for no other purpose than them-

[2]If the agent knows that the message to Dan is in reaction to the message from Chuck, it can, for example, be more helpful a week later when the user asks, "Did I ever react to the message from Chuck?"

[3]The focus act is what is called the "discourse segment purpose" in SharedPlan theory (see [16, 17]). The theory also specifies the rules by which discourse segments (contexts) are pushed and popped.

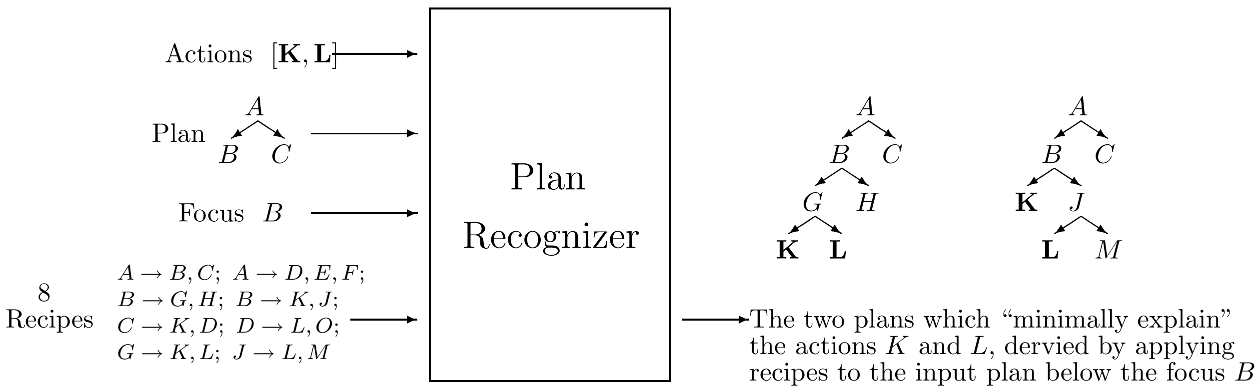


Figure 2. Simple example of inputs and outputs of plan recognition for collaboration

selves. Primitive actions can be executed directly, while non-primitive (abstract) actions are achieved indirectly by achieving other actions. We assume a predicate DONE?(A) which returns true if A has been achieved or executed. In our implementation, each action also has an associated type, parameters, timestamp, and so on; but will not need to explicitly refer to these here.

Beliefs concerning hierarchical goal decomposition (such as found in a SharedPlan) are formalized as a tuple $\langle \mathcal{A}, \mathcal{E}, \mathcal{C} \rangle$, which we will simply call a "plan," where $\mathcal{A}$ is a set of actions, $\mathcal{E}$ is a set of directed acyclic edges on $\mathcal{A}$, where the edge $A_i \rightarrow A_j$ means that A_j is a step in achieving A_i, and where $\mathcal{C}$ is a set of constraints. $\mathcal{C}$ may include temporal ordering between actions in $\mathcal{A}$, as well as other logical relations among their parameters.[4] As shown in Figure 2, plans can be viewed as trees (with an associated set of constraints, not shown in diagrams).

For plan $P = \langle \mathcal{A}, \mathcal{E}, \mathcal{C} \rangle$, we define $A_i \in P$ to mean $A_i \in \mathcal{A}$; we assume a predicate CONSISTENT?(P) which determines whether $\mathcal{C}$ is satisfiable; and we define a function REPLACE(P, A_1, A_2) which returns the (possibly inconsistent) plan resulting from replacing action A_1 with action A_2 in $\mathcal{A}$, $\mathcal{E}$, and $\mathcal{C}$.

Recipes are methods for decomposing non-primitive actions into subgoals. We represent recipes as functions that map an action to a plan that achieves the action. Let $\mathcal{RECIPE}$ be a set of recipes, where each recipe R_k is a function from a non-primitive action A_i to a plan $\langle \mathcal{A}', \mathcal{E}', \mathcal{C}' \rangle$, where $\mathcal{A}_i$ is in $\mathcal{A}'$ and for every $A_j \neq A_i$ in $\mathcal{A}'$ there is an edge $A_i \rightarrow A_j$ in $\mathcal{E}'$ i.e., the A_j are the *steps* in recipe R_k for achieving A_i.[5] For convenience, we define a function EXTEND(P, R_k, A_i), which returns the plan $\langle \mathcal{A} \cup \mathcal{A}', \mathcal{E} \cup \mathcal{E}', \mathcal{C} \cup \mathcal{C}' \rangle$.

2.1 Plan Recognition

As shown in Figure 2, the inputs to our plan recognizer are a sequence of actions $[A_1, ..., A_n]$, a plan $P = \langle \mathcal{A}, \mathcal{E}, \mathcal{C} \rangle$, a focus action $f \in P$, and a recipe library $\mathcal{R} \subseteq \mathcal{RECIPE}$. The output of the recognizer is a (possibly empty) set of extensions of P which "minimally explain" the input actions by applying recipes "below" the focus. More formally, each output plan, $P' = \langle \mathcal{A}', \mathcal{E}', \mathcal{C}' \rangle$, has the following properties:

[4] As shown in line 16 of Figure 1(c), our implementation allows recipes with optional steps, but for simplicity we do not include this feature in our formulation here.

[5] R_k may return the plan $\langle \{A_i\}, \emptyset, \emptyset \rangle$ if it is not applicable to A_i.

(a) Plan recognition.

```
RECOGNIZE([A_1,..,A_n],P,f,R) ≡
EXPL ← ∅, Q ← ∅
if P = ⟨∅,∅,∅⟩
  foreach T_i ∈ TOP
    add ⟨[A_1,..,A_n], ⟨{T_i},∅,∅⟩, T_i⟩ to Q
else foreach g_i ∈ FRINGE(P, f)
  add ⟨[A_1,..,A_n], P, g_i⟩ to Q
until Q = ∅
  remove ⟨[A'_1,..,A'_n'], P', act⟩ from Q
  P'' ← REPLACE(P', act, A'_1)
  if CONSISTENT?(P'')
    if n' = 1 add P'' to EXPL
    else foreach g_i ∈ FRINGE(P'', f)
      add ⟨[A'_2,..,A'_n'], P'', g_i⟩ to Q
  if act ∉ PRIM
    foreach recipe R_k ∈ R
      P''' ← EXTEND(P', R_k, act)
      foreach s_j, where act → s_j ∈ P'''
        add ⟨[A'_1,..,A'_n'], P''', s_j⟩ to Q
return EXPL
```

(b) Focus, ambiguity and clarification.

```
plan ← ⟨∅,∅,∅⟩, focus ← null, acts ← [ ]
repeat
  wait for next input action A_i,
  if DONE?(root of plan)
    plan ← ⟨∅,∅,∅⟩, focus ← null
  add A_i to acts
  pick ← null
  EXPL ← RECOGNIZE(acts, plan, focus, R)
  if EXPL = ∅
    set focus to root of plan
    EXPL ← RECOGNIZE(acts, plan, focus, R)
  if |EXPL| = 1
    remove pick from EXPL
  else if EXPL = ∅ or |acts| > MaxWait
    pick ← CLARIFY(EXPL)
  if pick ≠ null
    plan ← pick
    focus ← UDATEFOCUS(plan, A_i)
    acts ← [ ]
```

Figure 3. Pseudo-code for algorithms.

1. $\{A_1, ..., A_n\} \subseteq A'$,
2. every action in $(\mathcal{A}' - \mathcal{A})$ is reachable in $\mathcal{E}'$ from f,
3. P' can be derived from P by a composition of calls to EXTEND(..., R_k, ...), where $R_k \in \mathcal{R}$, and REPLACE(..., ..., A_k), where $A_k \in \{A_1, ..., A_n\}$, and
4. no smaller plan $\langle \mathcal{A}'', \mathcal{E}'', \mathcal{C}'' \rangle$, and $A'' \subseteq A'$, $E'' \subseteq E'$, $C'' \subseteq C'$, satisfies these properties.

Figure 3(a) shows pseudo-code for a simple plan recognizer that performs an exhaustive search of all possible ways of extending the input plan to explain the input actions. To understand the search space, consider how the input plan might be extended to include the first input action A_1. To explain A_1, the recognizer must apply some sequence of recipes $R_1, ..., R_k$ to the input plan P and then replace an action in the resulting plan with A_1.[6] The first recipe R_1 must be applied to a non-primitive action g_o in plan P that has not yet been expanded. Additionally, g_o must be beneath the focus act f in the subgoal structure of P. The function FRINGE(P, f) returns the set of actions in P reachable from f which are leaves of the plan tree and are not DONE?.

After applying recipe R_1 to an action g_o on the fringe of P, the recognizer only considers applying a second recipe, R_2, to g_o's subgoals, i.e., the steps added by R_1. This is justified because a plan can never be *minimally* extended to explain one action by applying recipes to multiple actions in the original plan. Similarly, the recognizer need only consider applying recipe R_3 to steps added by R_2 and, generally, only considers applying R_i to the steps added by R_{i-1}. It follows that the size of the search space to explain one action is bounded by $F(R \times S)^L$, where S is the maximum number of steps in a recipe, R is the maximum number of recipes applicable to an action, F is the number of actions on the fringe of P, and L is

[6]The recipe sequence can have length zero, i.e., we replace an action already in the plan with A_1.

the length of the longest sequence of recipes $R_1, ..., R_L$ the algorithm must consider.[7]

When the recognizer finds a plan P' which explains A_1 but there are more input actions to explain, it repeats the entire process to find all ways of extending P' to also explain action A_2, and so on until it has found all plans that minimally explain every input action. Since the algorithm recurses for each input action, its worst-case complexity is $\mathbf{O}((F'(R \times S)^L)^N)$, where F' is the maximum fringe size at any point and N is the number of input actions.

How does this compare to the general case? In the general case, the recognizer needs to search the entire plan space, because its output plans must contain no non-primitive steps that have not been decomposed. Let d be the depth of the deepest possible plan. Note that $d \geq L$, because if L recipes can be applied to explain a single action, then there must be a plan of at least depth L. The total number of plans that have to be searched is then $\mathbf{O}((R^S)^d)$. Thus if the number of input actions is small, the collaborative plan recognition problem is significantly more tractable than the general plan recognition problem. We guarantee that the number of input actions will be small with a policy that asks for clarification whenever the number of unexplained actions a threshold (described in next section).

2.2 Focus, Ambiguity and Clarification

We now discuss how to incorporate plan recognition into a collaborative agent (summarized in Figure 3(b)). It is beyond the scope of this paper to present the full SharedPlan discourse interpretation algorithm [18] used in Collagen. Instead, we concentrate on the role of the focus of attention and what to do when the recognizer returns multiple explanations.[8]

First, consider the focus of attention. In natural dialogue, people use context to help understand otherwise ambiguous statements. People do not typically even think about all possible interpretations of a potentially ambiguous phrase, such as "Let's save it", if there is an obvious interpretation that makes sense given what was just said. Analogously, we use the focus act to restrict the search space of our plan recognizer. Only if the recognizer fails to find any explanations using the current focus do we expand the context; we do so by setting the focus to the root of the current plan and calling the recognizer again. The function UPDATEFOCUS(*plan*, *act*) in Figure 3(b) returns *act* if *act* is not DONE?; otherwise *act*'s nearest ancestor in the plan that is not DONE?. Thus, the focus is in general set at the lowest-level goal which is not yet achieved, but includes the last observed or mentioned act.

Of course, the focus of attention does not guarantee a unique explanation. When ambiguity arises, we choose between two options: either wait or ask for clarification. A reason to wait is that future actions might resolve the ambiguity. A reason to ask now is that a collaborative system can be much more helpful when it knows what the user is doing. We believe it will be very difficult, in general, to compute the precise utility of asking for clarification. Instead, we use a simple heuristic: we ask for clarification as soon as there are $MaxWait$ or more unexplained actions, where $MaxWait$ is a small integer, currently set to 2.

We now briefly discuss how to manage a clarification sub-dialogue. Our collaborative agent first asks the user about the purpose of his most recent action, such as in lines 22–24

[7]In general, there might not be a bound on L due to cycles in the recipe set. In practice, we halt search whenever a cycle is encountered. For simplicity, here we assume that the recipe library is acyclic (as in [11]).

[8]There are also strategies in Collagen, not described here, for when the recognizer returns no explanations as well as methods for clarification of ambiguity between starting a new top level goal vs. working on optional steps of the current recipe.

(a) With and without plan recognition

	communications	executed actions	clarification questions
with rec.	0	5.2	1.2
w/o rec.	4.4	5.2	0

(b) Various tolerances for ambiguity

actions before clarify	questions asked per plan	steps per plan with ambiguity	CPU secs used by recognizer
1	2.6	0	.68
2	1.2	1.4	.94
3	.83	2.14	1.32

Figure 4. Experimental results averaged over 100 randomized trials in the email domain.

in Figure 1(c). If ambiguity remains, the agent then asks about the purpose of other actions in the user's plan which would disambiguate intermediate recipes. In general the agent can pursue a variety of clarification strategies, including potentially lengthy dialogues in which the agent asks the user many simple questions, brief dialogues which require the user to choose a possible explanation from the agent's current hypotheses, and a mixture of the two. At present, our agent engages in the potentially lengthy dialogues, but we intend to expand its repertoire to include other strategies.

3 Evaluation

We have incorporated plan recognition into Collagen. We now present results from preliminary, randomized experiments in the Lotus eSuite$^{\text{tm}}$ email domain, with the disclaimer that the performance of plan recognition is quite sensitive to the domain and the structure of the recipe library. As mentioned in Section 1.1, the recipe library was developed based on informal observation of real users. It currently contains 31 recipes, 32 primitive actions, and 19 non-primitive actions. These experiments are a first step in demonstrating and measuring the value of plan recognition in human-computer collaboration.

In each trial, we randomly instantiated a plan and simulated a user executing it. Without recognition, the user must communicate about every non-primitive action in the plan. With recognition, we chose to simulate an extreme case in which the user never volunteers information but instead executes the primitive actions and answers clarification questions from the agent. As shown in Figure 4(a), without recognition, the user has to communicate, on average, about 4.4 goals per plan. With recognition, the user only has to answer, on average, about 1.2 clarification questions per plan.

Recall that our algorithm has a parameter, $MaxWait$, which determines when it will ask for clarification. In the above experiments, $MaxWait$ was set to 2. Figure 4(b) shows results from experiments with different settings of $MaxWait$. Note that as $MaxWait$ increases, the number of clarifications decreases, but the recognizer knows what the user is doing less often and plan recognition requires more CPU time.

4 Related Work

The main difference between our work and previous work is the setting for plan recognition. Our work shows how to leverage properties of the collaborative setting to make plan recognition more tractable. We also demonstrate the value and practicality of plan recognition in an implemented collaborative system. Indeed, Collagen is now one a the few substantial systems with a plan recognizer that reasons over plans and goals.

The dominant framework for plan recognition research has been "keyhole" recognition, in which the observed actor is unaware of or indifferent to the observer (e.g.,[11]). Our recognizer takes two inputs that a keyhole recognizer does not: a partially elaborated plan and a focus act. These additional inputs simplify the plan recognition task because the recognizer must only extend the input plan, by applying recipes below the focus, just enough to explain the input actions. In the collaborative setting, the role of plan recognition is not to cleverly deduce an actor's plan, but rather to allow collaborators to communicate more naturally and efficiently.

In the proliferation of recent work on plan recognition, researchers have addressed various limitations of the keyhole framework. We now discuss a variety of these approaches and illustrate how their settings and resulting techniques differ from our own.

Vilain [23] presented a plan recognizer that runs in polynomial time. However, his algorithm only recognizes top level goals which is not sufficient for our purposes, and can only achieve polynomial time if the steps in the recipes are totally ordered, which is too restrictive for our domains. In general, however, we believe Vilain's or other's fast recognition algorithms (e.g., [13, 14]) could be adapted to our formulation.

Lochbaum [15] presented a plan recognition algorithm based on the SharedPlan model of collaboration. Her plan recognizer does not chain recipes together, as our does, and thus performs only "one level deep" recognition. It does, however, make use of a wider range of relations by which actions contribute to goals than we do.

Plan recognition has also been studied within a collaborative setting in which each participant works on their own plan but pools information and coordinates actions with others [10, 21]. In particular, this work explores the opportunity for plan recognition when a participant announces that one of their goals has failed.

Our work is close in spirit to research on plan recognition for cooperative dialogues. Our use of context to narrow the scope of plan recognition resembles Carberry's focusing heuristics [3]. Much work on cooperative response generation addresses the listener's need to know only enough of the speaker's plan to answer her questions adequately (e.g., [1, 4]). In contrast, we concentrate on a collaborative setting in which a joint plan is maintained by all collaborators and there is a shared artifact that all participants can interact with. A related distinction concerns ambiguity. The primary source of ambiguity in the cooperative response generation work resides in determining the user's top level goal (e.g., does the student want to take the course for credit or as an audit— see [12]). In our work, ambiguity arises because there are multiple ways of connecting actions to known higher level goals.

A variety of strategies for reducing ambiguity have been proposed. These include adopting the worst possible explanation for the observer in adversarial settings [22], and assuming the observed person is doing what a expert system would suggest in a medical assistance domain [6]. In our collaborative setting, we use the focus of attention to reduce ambiguity, but failing this, we believe it often best just to ask the person what they are doing.

Applying probabilistic approaches to recognition (e.g., [2]) in collaboration would likely be beneficial. However, we do not believe this will eliminate ambiguity or the need for clarification in human-computer collaboration because both seem fundamental to human-human collaboration.

Plan recognition has often been proposed to facilitate intelligent user help (e.g., [7, 13]).

Typically, the computer watches the user "over the shoulder" and jumps in with advice or assistance when the recognizer deduces the user's goals (e.g., [24]). This approach does not view human-computer interaction as collaboration, in which all participants are committed to maintaining mutual understanding of the common goals. Instead, it makes the the (to us) implausible assumption that it is possible to infer the user's goals and plans by observing only primitive interface actions and to choose appropriate assistance without any mutual understanding.

5 Conclusion

Human-computer collaboration is a fruitful application for plan recognition because all participants are committed to maintaining a mutual understanding of the goals and actions to be performed. The question isn't whether the software agent will know the user's plan, but how the agent and the user can best communicate their intentions to each other. We have shown that plan recognition can allow more efficient and natural communication between collaborators, and can do so with relatively modest computation effort.

References

[1] L. Ardissono and D. Sestero. Using dynamic user models in the recognition of the plans of the user. In *User Modeling and User Adapted Interaction*, volume 2, pages 157–190, 1996.

[2] M. Bauer, S. Biundo, D. Dengler, J. Kohler, and Paul G. PHI–a logic-based tool for intelligent help systems. In *Proc. 13th Int. Joint Conf. AI.* 1993.

[3] S. Carberry. Incorporating default inferences into plan recognition. In *Proc. 8th Nat. Conf. AI*, volume 1, pages 471–8, july 1990.

[4] R. Cohen, K. Schmidt, and P van Beek. A framework for soliciting clarification from users during plan recognition. In *Proceedings of the Fourth Int. Conference on User Modeling*, pages 11–17, 1994.

[5] G. Ferguson and J. Allen. Trips: An integrated intelligent problem-solving assistant. In *Proc. 15th Nat. Conf. AI*, pages 567–572, 1998.

[6] A. Gertner and B. Webber. A bias towards relevance: Recognizing plans where goal minimization fails. In *Proc. 13th Nat. Conf. AI*, pages 1133–1138, 1996.

[7] B. Goodman and D. Litman. Plan recognition for intelligent interfaces. In *Proc. 6th IEEE Conf. AI Applications*, 1990.

[8] B. J. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, October 1996.

[9] B. J. Grosz and C. L. Sidner. Plans for discourse. In P. R. Cohen, J. L. Morgan, and M. E. Pollack, editors, *Intentions and Communication*, pages 417–444. MIT Press, Cambridge, MA, 1990.

[10] Curry I. Guinn. Mechanisms for dynamically changing initiative in human-computer collaborative discourse. In *Human Interaction with Complex Systems Symposium*, 1996.

[11] H.A. Kautz and J.F. Allen. Generalized plan recognition. In *Proc. 5th Nat. Conf. AI*, pages 32–37, 1986.

[12] L. Lambert and S. Carberry. A tripartite plan-based model of dialogue. In *Proc. 29th Annual Meeting of the ACL*, pages 47–54, Berkeley, CA, 1991.

[13] N. Lesh and O. Etzioni. A sound and fast goal recognizer. In *Proc. 14th Int. Joint Conf. AI*, pages 1704–1710, 1995.

[14] D. Lin and R. Goebel. A message passing algorithm for plan recognition. In *Proc. 12th Int. Joint Conf. AI*, volume 1, pages 280–5, July 1990.

[15] K. E. Lochbaum. An algorithm for plan recognition in collaborative discourse. In *Proc. 29th Annual Meeting of the ACL*, Berkeley, CA, 1991.

[16] K. E. Lochbaum. Using collaborative plans to model the intentional structure of discourse. Technical Report TR-25-94, Harvard Univ., Ctr. for Res. in Computing Tech., 1994. PhD thesis.

[17] K. E. Lochbaum. A collaborative planning model of intentional structure. *Computational Linguistics*, 1998. Forthcoming.

[18] K. E. Lochbaum, B. J. Grosz, and C. L. Sidner. Models of plans to support communication: An initial report. In *Proc. 8th National Conf. on Artificial Intelligence*, pages 485–490, Boston, MA, July 1990.

[19] C. Rich and C. Sidner. COLLAGEN: A collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction*, 1998. Special issue on Computational Models of Mixed-Initiative Interaction. Forthcoming. Also published as MERL Technical Report 97-21a.

[20] J. Rickel and W. L. Johnson. Animated agents for procedural training in virtual reality: Perception, cognition, and motor control. *to appear in Applied Artificial Intelligence*, 1998.

[21] Ronnie W. Smith, D. Richard Hipp, and Alan W. Biermann. A dialog control algorithm and its performance. In *Third Conference on Applied Natural Language Processing*, April 1992.

[22] Milind Tambe and Paul Rosenbloom. RESC: An approach for real-time, dynamic agent-tracking. In *Proc. 14th Int. Joint Conf. AI*, pages 103–110, 1995.

[23] M. Vilain. Getting serious about parsing plans: A grammatical analysis of plan recognition. In *Proc. 8th Nat. Conf. AI*, pages 190–197, 1990.

[24] Robert Wilensky, David Chin, Marc Luria, James Martin, James Mayfield, and Dekai Wu. The Berkeley UNIX Consultant project. *Computational Linguistics*, 14(4):35–84, 1988.