5 Plan Recognition: Background

In Chapter 1, we outlined several key areas in which progress must be to support agent-based dialogue systems. First we mentioned that we needed a dialogue model as well as a way of describing the communicative intentions associated with utterances. We have presented solutions to both of these in Chapter 4. Once we have a representation of communicative intentions for utterances, we need a way of performing *intention recognition*: the recognition of communicative intentions based on context and the speaker's utterance.

Unfortunately, a full model for intention recognition in agent-based dialogue is beyond the scope of this thesis. Instead, in the remaining chapters, we make several contributions to the more general field of plan recognition which we believe are the first steps towards creating a practical intention recognizer for agent-based dialogue.

Although much work has been done in intention recognition (see [Carberry1990b; Lochbaum, Grosz, and Sidner2000]) these methods assume a plan-based model of dialogue and are not directly applicable to our agent-based model. We do believe, however, they can be extended to our model, so we do not discount them. Instead, we focus on several problems with intention recognition and the more general problem of plan recognition. As plan recognition is a more general form of intention recognition, solutions in the general domain will be applicable to current intention recognizers as well

as future work on intention recognition for agent-based dialogue.

In this chapter, we first discuss the relationship between intention recognition and plan recognition. We then outline some general requirements for plan recognition and discuss previous work in this field. Finally, we conclude and introduce the solutions presented in the coming chapters.

5.1 Intention Recognition and Plan Recognition

Intention recognition is a special case of *plan recognition*: the general task of inferring an agent's goals and plans based on observed actions. In intention recognition, observed actions are speaker utterances and the goals are the speaker's communicative intentions.

Plan recognition is typically divided into two types. In *keyhole recognition*, the agent being observed is unaware of (or does not care about) the observation. In *intended recognition*, on the other hand, the agent knows it is being observed and chooses its actions in a way such to make its plan clear to the observer.[1] Intention recognition is a type of *intended recognition*, as the speaker forms his actions (utterances) in such a way to make his communicative intentions clear to the hearer.[2]

Despite the fact that intention recognition is a type of intended recognition, and that the speaker forms his utterances intentions so as to make his communicative intentions "easy" to recognize, intention recognition remains a hard problem for the community, both in terms of domain-independence, as well as runtime efficiency. All intention recognizers that we are aware of use at their core a plan recognizer. Thus any problems

[1] A third type of plan recognition occurs when the agent is trying to thwart recognition of its plans. Pollack [1986] calls this an *actively non-cooperating actor*. Very little research has been done for this third type of recognition (although cf. [Azarewicz et al.1986]), which may be why it is frequently not included in the typology.

[2] Note, that this is the case even in deceptive conversation, as the speaker forms his utterances so as to make his feigned intentions clear.

with plan recognizers in general have been inherited also by intention recognizers. We now turn our attention to plan recognition in general.

5.2 Requirements for Plan Recognition

Plan recognition has not only been used in dialogue systems, but also in a number of other applications, including including intelligent user interfaces [Bauer and Paul1993; Horvitz and Paek1999; Rich, Sidner, and Lesh2001], traffic monitoring [Pynadath and Wellman1995], and hacker intrusion detection [Geib and Goldman2001]. All of these applications (including dialogue) have a common set of requirements they place on a plan recognizer:

1. **Speed:** Most applications use plan recognition "online," meaning they use recognition results before the observed agent has completed its activity. Ideally, plan recognition should take a fraction of the time it takes for the observed agent to execute its next action.

2. **Early prediction:** In a similar vein, applications need accurate plan prediction as early as possible in the observed agent's task execution. Even if a recognizer is fast computationally, if it is unable to predict the plan until after it has seen the last action in the agent's task, it will not be suitable for online applications, which need recognition results *during* task execution.

3. **Partial prediction:** If full recognition is not immediately available, applications can often make use of partial information. For example, if the parameter values are not known, just knowing the goal schema may be enough for an application to notice that a hacker is trying to break into a network.

As we discuss below, previous work in plan recognition does not provide these needed features. Typically, systems will sacrifice one attribute for another.

5.3 Previous Work in Plan Recognition

In this section, we discuss previous work in plan recognition. This can be divided into two different types. The first is plan recognition based on logic, while the second includes probabilities.

5.3.1 Logic-based Plan Recognition

Most plan recognizers use a plan library, which represents goals in the domain, and the (typically hierarchical) plans associated with them. Logic-based recognizers can be characterized by the use of logical methods to exclude goals and plans in the hierarchy made impossible given the observed actions.

There have been several types of logic-based plan recognizers. We first discuss work that bases plan recognition on chaining. Then we discuss plan recognition as circumscription, and finally, plan recognition based on parsing algorithms.

Plan Recognition as Chaining

Allen and Perrault [1980] created one of the earliest plan recognizers. Given a single observed action, the recognizer used various rules to either forward chain from the action to a goal, or backwards chain from an expected goal to the action. Rules supported not only chaining on preconditions and effects, but also hierarchically to higher levels of recipes. Heuristics were used to control and focus rule application for chaining.

Carberry [1983; 1990b] extended Allen and Perrault's work to cover multiple successive action observations. Each new action is independently upwards chained until further chaining would create ambiguity. Then, the new action is merged into the plan recognized so far based on previous observations. Ambiguity of where a plan "attaches" is resolved by the use of focusing heuristics, which assume that action observations are often coherently clustered together.

Plan Recognition as Circumscription

The seminal work on plan recognition was done by Kautz [1987; 1990; 1991][Kautz and Allen1986], who casts plan recognition as the logical inference process of circumscription. This provided a rich plan representation — essentially that of first order logic and a temporal logic to represent actions and time.

Kautz represented the space of possible plans as a plan library called the event hierarchy, which included both abstraction and decomposition (subaction) relations. Goals and actions were represented as complex schemas that included parameter values. Certain actions were labeled as *end* actions, meaning that they were an end unto themselves, or a possible ultimate goal of an agent.

Kautz showed that by assuming that the event hierarchy is complete and that all events are disjoint, plan recognition becomes a problem of logical circumscription. Given a certain set of observations (also represented in first order logic) a set of *covering models* is computed which are somewhat like possible worlds in which the observations are true, and each contains a separate possible goal and plan for the agent.

Runtime of the recognizer is exponential in the size of the event hierarchy (e.g., all goals and subgoals), which means it is not scalable to larger, more realistic domains. However, it does have several other features, including a rich representational power (including interleaved plans, partially-ordered recipes, and goal and action parameters, to name a few). It also supports partial prediction through the ability to predict just a goal schema as well as to predict an abstract goal. As discussed below, it also suffered from the general inability of logic-based systems to deal with ambiguity.

Plan Recognition as Parsing

To make plan recognition more tractable, Vilain [1990] describes a method of converting a subset of Kautz's plan hierarchy into a grammar. Plan recognition is then performed by running a chart parser over observed actions. By using this approach,

runtime complexity becomes $O(|H|^2 n^3)$ where H is the set of goals and subgoals in the plan library and n is the number of observed actions.

This vast improvement over exponential runtime comes at a cost: the grammar approach decreases the representational power substantially; it requires totally-ordered recipes; and does not handle goal and action parameters. (Vilain suggests that parameters could be handled as a feature grammar, although this would make the algorithm NP-Complete.) The lack of goal parameters means a possible explosion in the number of goals in certain domains, since each instance of a goal schema must be modeled as a separate goal.

In addition, it is not entirely clear if online predictions can be made by the recognizer. Vilain suggests that this could be done by looking at dotted rules on the chart, but it is not clear how much predictive power this would give the recognizer.

General Shortcomings of Logic-based Recognizers

A general problem with logic-based recognizers (as noted in [Charniak and Goldman1993]) is their inability to deal with ambiguity. This comes from the fact that, upon each new observed action, they prune away only the predictions which become logically impossible. Unfortunately, this impacts early prediction substantially, as most plan recognition domains are highly ambiguous, especially when only the first few actions from a plan have been observed. In order to disambiguate further, uncertain reasoning is often used.

5.3.2 Probabilistic Plan Recognition

Several lines of probabilistic plan recognition have been explored. We discuss here the use of Dempster-Shafer theory, probabilistic abduction and belief networks.

Dempster-Shafer Theory

Two systems use Dempster-Shafer theory (DST) to add probabilistic reasoning to plan recognition. Carberry [1990a] used DST to her logic-based recognizer (see above) to do default inferencing when further upwards chaining was ambiguous.

Bauer [1995] uses DST to represent and combine the probability of goals given observed actions. He uses a subset of Kautz' plan library which includes an abstraction hierarchy and a single level partially ordered recipes. He uses results of previous recognition sessions to learn a DST *basic probability assignment* (bpa) which roughly corresponds to the a priori goal probability (an abstract goal is defined a the set of its base goals).

In addition, he uses the plan library itself (and a corpus if available [Bauer1994]) to train another set of bpas which roughly correspond to the probability of a goal given an observed action. A bpa is defined for each action in the domain, and gives probability mass to each goal in which it is part of a recipe.

The recognition algorithm is as follows: the prediction bpa is initialized to the a priori goal probabilities. Then, for each observed action, the precomputed bpa for that action is retrieved, and then each of the possible goals is logically checked with respect to constraints (e.g., ordering constraints). If all constraints for all goals hold, the bpa remains the same. Otherwise, probability mass is taken away from logically impossible goals and redistributed. Then, this bpa is combined with the prediction bpa by Dempster's rule of combination resulting in the new prediction bpa.

It is unclear if this algorithm is scalable, however. DST is known to be exponential in the general case, and although Bauer mentions some possible solutions (like restricting bpa subsets to be only abstract goals) it is unclear how this restriction would be handled by Dempster's rule of combination and what the effect on recognition would be. Also, the approach does not support a decomposition hierarchy, and thus is unable to make predictions about intermediate subgoals and plans.

Probabilistic Abduction

Appelt and Pollack [1991] designed a framework in which plan recognition[3] could be modeled as weighted abduction. The framework allows inferences to be encoded as prolog-like rules with a weight attached to them. If the consequent of the rule can be logically proven, there is no cost. However, if it is assumed, then the algorithm incurs the cost of the weight of that step. Out of all possible solutions, the one with the lowest weight is then chosen.

Appelt and Pollack mention several drawbacks to their work. First, in the general case, the algorithm is intractable (NP-hard). Also, weights assigned to abduction rules are not probabilities and must be assigned by hand. They report that local changes in these rules can affect global recognition in subtle ways.

Goldman et al. [1999] also model plan recognition as an abduction problem. They model the process of plan execution, and then reverse the decisions to make an abductive model. In addition, three parts of the execution process are made probabilistic: the agent's choice of a top goal, the agent's choice between competing recipes for a goal (or subgoal), and the agent's choice of what action to execute next (from the set of currently executable actions).

This framework is the first of which we are aware to model plan recognition with the fact in mind that the agent is executing the action, as opposed to other work which just works on a plan library data structure. Because of this, they are able to model many things with other systems could not, including multiple, interleaved plans and evidence from *failure* to observe an action.

Like Appelt and Pollack, however, Goldman et al. define a theoretical framework, but do not deal with the problem of tractability. Although they do not analyze complex-

[3] Actually, they do what they call *plan ascription*, which is the (more difficult) process of attributing mental states to an agent, the combination of which can then signal that the agent has a mental plan of the form described in [Pollack1986].

ity, it is likely that this framework suffers from the same intractability problems that Appelt and Pollack's abduction framework had.

Belief Networks

Charniak and Goldman [1993] use a belief network (BN) to encode the plan recognition problem. Nodes in their BN include propositions such as the existence of an object or event, its type, and its role within some plan. As actions are observed, they are added to the network in this kind of encoding (with the appropriate arcs between them), and new nodes are generated which explain possible connections between them and the possible plans encoded in the network. After these nodes (and connections) have been added, the posteriori probabilities of other nodes (especially goals) can be computed to predict the plan.

Huber et al. [1994] propose a method to automatically convert a plan execution library into a BN, albeit one with a different structure. Their BNs include only events (not parameters) and directly encode the links between them (not through intermediary role nodes like Charniak and Goldman).

Unfortunately, reasoning with BNs is exponential in the size of the network. To attempt to deal with this, Charniak and Goldman use a message-passing algorithm to keep the number of nodes restricted, although the size of the network grows with each new observation (and the likely goals chained from it). The system of Huber et al. has a static BN and is likely not scalable to large plan libraries.

5.4 Goal Recognition

In the last section, we discussed previous work in plan recognition. We now discuss work on a special case of plan recognition: *goal recognition*. Whereas the task of plan

recognition is the recognition of an agent's goal and plan, goal recognition attempts only to recognize the goal.

Although not as informative as full plan recognition, goal recognition has been an active research area of late, partially because it has been noticed that many applications simply do not need full plan recognition results. For example, Horvitz and Paek [1999] built an AI receptionist which observed actions (including natural language utterances) to determine the user's goal, which the receptionist then did for them. Here, the goal was something only the receptionist itself could accomplish, thus the users typically did not have a plan.

Additionally, goal recognition naturally removes some of the ambiguity present in plan recognition. It is still the case that a set of observed actions could be accounted for by any number of goals, but plan recognition has the additional ambiguity that, even if the agent's goal can be unambiguously identified, it could be associated with a large number of plans, all consistent with the observed actions. For this reason, in fact, most of the plan recognizers mentioned above do not predict a fully-specified, fully-disambiguated plan at each timestep, but rather a *partial* plan that includes only those parts which are disambiguated. We believe that a fast goal recognizer could be used in a hybrid system to focus the search in a slower plan recognizer (although we leave this to future work).

Goal recognizers can be classified by the goal structure they try to recognize. *Flat* goal recognizers attempt to recognize goals at just one level, typically the top-level goal. *Hierarchical* goal recognizers, on the other hand, attempt to recognize active subgoals in addition to the top-level goal. Note that hierarchical goal recognition is different from general plan recognition in that in plan recognition, the attempt is to recognize the entire plan tree, whereas with hierarchical goal recognition, one only attempts to recognize the chain of the active subgoals, i.e., the line of subgoals which trace the last observed action to the top-level goal.

We first discuss previous work on flat goal recognizers, and then hierarchical goal

recognizers.

5.4.1 Flat Goal Recognizers

Logic-based Systems

Following recent successful work on using graph analysis in doing planning synthesis [Blum and Furst1997], Hong [2001] uses graph analysis for goal recognition. His system incrementally constructs a *goal graph* consisting of nodes representing state predicates and observed actions. Each observed action has incoming edges from state predicates that fulfill its preconditions, and outgoing edges to predicates that are its effects. Predicates which remain true across actions are also connected. Predicates also connect to goal nodes whose goal state they contribute to. This provides a list of all goal states partially or fully fulfilled by the actions up until the last observation. The algorithm then uses the graph to compute which goals were causally linked to which actions. If a majority of observed actions contributed to a certain goal, it is reported as a recognized goal.

The algorithm does not require a hand-built plan library, but rather just uses descriptions of base-level actions and high-level goal states. As Hong points out, however, this algorithm is only appropriate for post hoc goal analysis, and not online goal recognition, as it does not quickly converge on a single goal. The reason for this is that the effects of an action may contribute to any number of goals, and it only becomes clear near the end of the agent's execution which of these is really being focused on.

Lesh's RIGS-L system [Lesh and Etzioni1995b; Lesh and Etzioni1995a; Lesh and Etzioni1996; Lesh1998] uses analysis of a different kind of graph to do goal recognition. RIGS is initialized with a fully-connected *consistency graph* of action and goal schemas and instantiated actions observed thus far. Edges between action schema nodes are used to signify *support* between them, and edges to goal schema nodes signify *completion*. Given this graph, the algorithm uses rules to remove graph elements while

still keeping the graph correct. For example, the *matching* rule removes an edge $e_{x,y}$ where no effect of x matches a precontidion of y (and thus does not directly support it). The *goal connection* rule deletes goal schemas which are no longer connected to the graph. After the algorithm has run, any goal schema that is not connected is no longer consistent with the evidence, and any remaining goal schemas are instantiated by the algorithm and predicted as possible goals.

The runtime complexity of RIGS-L is $O(|G| + (|A| + |L|)^6)$ where G is the set of goal schemas, A is the set of action schemas, and L is the set of observed actions. Note that, although this is linear in the number of goal schemas, it is only polynomial overall, unless $|G| >> |A|$, which we do not believe is the case in most domains.

Lesh then uses RIGS-L as a component of the BOCES goal recognizer, which uses *version spaces* from the machine learning field to represent the set of possible goals and mark which are consistent (without, however, actually *enumerating* the goals). The set of goals are defined and then based on this definition, the goal recognizer keeps track of boundaries between those goals which are consistent and which are not. Lesh shows that BOCES has a runtime complexity of $O(log(|G|))$ for a certain subclass of goals called *decomposable goals*, goals in which adding a conjunct makes them more specific (like searching for an item with a set of features). Runtime for other classes of goals is the same as that of RIGS-L.

For decomposable goals, BOCES has been shown to run quickly for even hundreds of thousands of goals. However, these goals are defined in a certain way, namely the combination of conjuncted domain predicates, which is typically the case in decomposable goals such as constrained searching. However, many typical goal recognition domains do not exclusively include decomposable goals. For decomposable goals, however, BOCES is probably unbeatable.

Logic-based goal recognizers in general also have the same drawbacks mentioned for logical plan recognizers above, namely, that they are unable to distinguish between logically consistent goals, which leads us to probabilistic flat goal recognizers.

Probabilistic Systems

Horvitz and Paek [1999] use a 3-layered Belief Network to recognize users' goals in a secretarial setting. The system not only uses observed actions in the network, but also other factors like world state. The top layer network tries to recognize an abstract goal. When confidence in a single goal at this level is high enough, control passes to the next level, which attempts to recognize a more concrete goal, and so on. The system is able to perform partial recognition because it can return just an abstract goal when it is not certain enough about a more specific version. As the system uses a Belief Network, its worst-case complexity is exponential in the size of the network. Also, as is the case for probabilistic systems, probability distributions must somehow be estimated for each of the nodes and it is unclear this would be done.

Albrecht et al. [1998] use a dynamic belief network (DBN) to predict the top-level goal and next action in a multi-user dungeon (MUD) game. They estimate probabilities from logs of actual game sessions, where a user attempts to complete one of 20 quests (goals). Although not reported, the runtime complexity of the recognizer appears to be linear in the number of goals, and is quite similar to the statistical goal schema recognizer we present in Chapter 7 (although see Section 7.2.1 for a discussion of differences). Their recognizer, however only recognizes atomic goals and is not able to handle parameters. It also does not support partial prediction. However, it was the first goal recognizer of which we are aware which used a large corpus to learn probabilities as well as to evaluate the recognizer.

5.4.2 Hierarchical Goal Recognizers

The last section discussed flat goal recognizers, which only recognize the agent's top-level goal. In this section, we report on several recent recognizers which recognize all of an agent's active subgoals, as well as the top-level goal.

Pynadath [1999][Pynadath and Wellman2000] uses probabilistic state-dependent

grammars (PSDGs) to do plan recognition. PSDGs are probabilistic context-free grammars (PCFGs) in which the probability of a production is a function of the current state. This allows, for example, the probability of a recipe (production) to become zero if one of its preconditions does not hold. Subgoals are modeled as non-terminals in the grammar, and recipes are productions which map those non-terminals into an ordered list of non-terminals or terminals. During recognition, the recognizer keeps track of only the current productions and the state variables as a DBN with a special update algorithm. The most likely string of current productions is predicted as the current hierarchical goal structure.

If the total state is observable, Pynadath claims the complexity of the update algorithm to be linear in the size of the plan hierarchy (number of productions).[4] However, if the state is only partially observable, the runtime complexity is quadratic in the number of states consistent with observation, which grows exponentially with the number of unobservable state nodes.

Additionally, the recognizer only recognizes atomic goals and does not take parameters into account. Finally, although the PSDG allows fine probability differences for productions depending on the state, it is unclear how such probability functions could be learned from a corpus, as the state space can be quite large.

Bui [2002][Bui, Venkatesh, and West2002] performs hierarchical recognition of Markov Decision Processes. He models these using an Abstract Hidden Markov Model (AHMM) which are multi-level Hidden Markov Models where a policy at a higher level transfers control to a lower level until the lower level 'terminates.' The addition of memory to these models [Bui2003] makes them very similar to the PSDGs used by Pynadath in that each policy invokes a 'recipe' of lower-level policy and does not continue until the lower level terminates.

Recognition is done using a DBN, but because this is intractable, Bui uses a method called Rao-Blackwellization (RB) to split network variables into two groups. The first

[4]This claim is disputed in [Bui2002].

group (which includes the state variables as well as a variable which describes the highest terminating state in the hierarchy) is estimated using sampling methods. Then, using those estimates, exact inference is performed on the second part (the policy variables). The separation is such that exact inference on the second group becomes tractable, given that the first group is known.

The recognizer was used in a system which tracked human behavior in an office building at three abstract levels, representing individual offices at the bottom level, then office groups, then finally the entire building. Policies at each level were defined specific to each region (for example the policy (behavior) of using the printer in the printer room). In this model, only certain policies are valid in a given state (location), which helps reduce the ambiguity. Typically, the domain is modeled such that lower-level policies become impossible as the agent moves to another room, which makes it fairly clear when they then terminate.

Although the algorithm was successful for this tracking task, it is unclear, however, how effective estimation of policy termination would be in general (e.g., when most policies are valid in most states). Also, similar to Pynadath, this method only recognizes atomic goals and does not support parameters.

5.5 Towards Statistical Goal Recognition

As mentioned above, we need goal recognizers which are fast, and make early (and possibly partial) predictions. However, most current recognizers are either not scalable or severely limit the representation of the domain.

In the following chapters, we present a *statistical goal recognizer* which uses machine learning techniques to train the recognizer on a particular domain given a corpus. As it learns domain behavior from the corpus, it does not utilize a plan library and does not therefore limit plan representation in that respect. In addition, it supports parameterized goal and action schemas and can make partial predictions if not all parameter

values are known. We will show that it is scalable and can make quick and early predictions.

The remainder of the thesis is as follows. As the recognizer needs a corpus to be trained on, in Chapter 6 we present the two corpora which we use in our experiments. The first was gathered from human users in the Linux domain. However, as many domains do not lend themselves to easy observation, we present a general method for stochastically producing artificial corpora for plan recognition and use this method to produce a corpus in the emergency planning domain.

In Chapter 7, we present a flat goal recognizer which is linear in the number of goals and present its performance on the two corpora described above. Finally, in Chapter 8, we extend this flat recognizer into a hierarchical goal recognizer and present experimental results for it as well.

Finally, in Chapter 9, we conclude the thesis and discuss directions of future work.