Recognizing Instantiated Goals using Statistical Methods

Nate Blaylock
Saarland University
Saarbrücken, Germany
blaylock@coli.uni-sb.de

James Allen
University of Rochester
Rochester, New York, USA
james@cs.rochester.edu

Abstract

We present our work on using statistical, corpus-based machine learning techniques to perform *instantiated goal recognition* — recognition of a goal schema and its parameter values. The recognizer is fast (linear in the number of goal schemas and observed actions) and is able to make partial predictions by optionally predicting individual parameter values for a goal schema. This allows it to make more accurate predictions after observing fewer actions than all-or-nothing prediction.

	Linux	**Monroe**
Total Sessions	457	5000
Goal Schemas	19	10
Action Schemas	43	30
Ave Actions/Session	6.1	9.5
Subgoal Schemas	N/A	28
Ave Subgoal Depth	N/A	3.8
Max Subgoal Depth	N/A	9

Table 1: Plan Corpora Statistics

1 Introduction

Much work has been done over the years in *plan recognition* which is the task of inferring an agent's goal and plan based on observed actions. *Goal recognition* is a special case of plan recognition in which only the goal is recognized. Goal and plan recognition have been used in a variety of applications including intelligent user interfaces [Bauer and Paul, 1993; Horvitz and Paek, 1999; Lesh *et al.*, 1999], traffic monitoring [Pynadath and Wellman, 1995], and dialogue systems [Carberry, 1990; Allen *et al.*, 2001].

For most applications, there are several properties required in order for goal recognition to be useful:

1. **Speed:** Most applications use goal recognition "online", meaning they use recognition results before the observed agent has completed its activity. Ideally, goal recognition should take a fraction of the time it takes for the observed agent to execute its next action.
2. **Early/partial prediction:** In a similar vein, applications need accurate goal 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 goal until after it has seen the last action in the agent's task, it will not be suitable for applications which need recognition results *during* task execution. If full recognition is not immediately available, applications can often make use of partial information.

In this paper, we model goals as parameterized action schemas[1] (see examples in Figures 2 and 3). We apply supervised machine-learning techniques to the task of *instantiated goal recognition*: the recognition of a goal schema and its parameter values.

Our recognizer has two nice features (which correspond to the two desired traits of goal recognizers described above). First, recognition is fast and scalable, running in time linear to the number of possible goal schemas and the number of observed actions. Second, the recognizer supports partial goal recognition, allowing it to make predictions earlier in the agent's task execution. It is able to predict just a goal schema or a goal schema and only a subset of its parameter values. This allows the recognizer to make predictions much earlier and more accurately than a strict full-prediction system.

The remainder of the paper is as follows: Section 2 briefly describes the plan corpora we use for training and testing. Section 3 describes our goal recognizer and experimental results. In Section 4, we discuss related work and in Section 5, we conclude and mention future work.

2 Plan Corpora

The first thing needed to use a machine-learning approach is data. A *plan corpus* is a set of *plan sessions* — collections of observed actions from an agent. For our approach, we need plan sessions that are labeled with the top-level goal the agent was pursuing during the session.

We use two different goal-labeled plan corpora to train and test our recognizer: the Linux corpus and the Monroe corpus. Statistics for the corpora are shown in Table 1.

[1] Note that this is different from another common representation, where goals are represented as conjunctions of state predicates.

2.1 The Linux Corpus

We collected the Linux corpus [Blaylock and Allen, 2004] from human Linux users at the University of Rochester.[2] In each session, a user was given an (English) description of a Linux task (a goal) and was instructed to solve it using any Linux commands (with a few rules such as no pipes, no `awk`, etc.) The users' commands and their results were recorded. Also, as we had no automatic means to detect goal completion, we required users, at the end of a session, to declare whether they had successfully completed the task.

Table 2 shows several of the goal schemas in the Linux corpus. We prefix parameter names with a dollar sign ($) to distinguish them from instantiated parameter values.

Post-Processing

In creating the final corpus, we performed post-processing to transform the raw corpus (of Linux commands and results) into a planning representation of the goals and actions.

First, we excluded all sessions which were reported as failures, as well as sessions with no valid commands. Although such data could possibly be useful for training a recognizer to recognize goals which will not be accomplished by the user alone, we decided to leave such research for future work.

We also converted issued Linux commands into parameterized actions. Unlike actions in many domains used in plan recognition, Linux commands do not nicely map onto a simple set of schemas and parameters, as we discuss in more detail below. To do the mapping, we defined action schemas for the 43 valid Linux command types appearing in the corpus. This allowed us to discard mistyped commands as well as many commands that resulted in errors.

Discussion

As discussed above, the Linux corpus was gathered semi-automatically from humans. As a consequence, it contains mistakes. A frequent mistake was typographical errors. The post-processing step described above helped ameliorate this somewhat — as it was able to detect incorrectly typed commands (at least in cases where the mistyped command wasn't also a successful command). However, it only checked the command itself, and not its parameter values. This lead to cases of the user using unintended parameters (e.g., `ls flie` instead of `ls file`), which affected parameter recognition as described below.

Another phenomenon that we were not able to automatically detect was the user's lack of knowledge about commands. For example, one user, upon getting the task of finding a file with a certain name tried several times in vain to use the command `grep` to do so, where the command he was likely looking for was `find`.[3] This resulted in a red herring for the recognizer, which, for the remainder of the session, predicted that the user was trying to locate text within a certain file.

Finally, another source of noise in the corpus is that the users themselves reported whether they had accomplished tasks successfully. We have seen several cases in the corpus where a user apparently misunderstood the task and reported success where he had actually failed. Overall, however, this does not appear to have happened very often.

[2]The Linux corpus is modeled after Lesh's Unix corpus [Lesh, 1998].

[3]The command `grep` is used to find text in a file or set of files, not to find a file in a directory tree.

2.2 The Monroe Corpus

As human data collection is expensive (and oftentimes not possible), we have developed a method of stochastically generating artificial plan corpora using an AI planner [Blaylock and Allen, 2005]. The general method is to model a domain plan library and then use a randomized planner to create plans given stochastically generated goals and start states.

Using this method, we created the Monroe corpus, which is an emergency response domain based on a similar domain for a dialogue system [Stent, 2000]. Examples of goal schemas in the corpus are shown in Table 3.

Table 1 shows a comparison of the contents of the Monroe corpus and the (post-processed) Linux corpus. The Monroe corpus consists of 5000 plan sessions with an average of 9.5 actions per session. The number of total sessions was, of course, artificially set and could have easily been changed. The 5000 sessions were generated on a high-end desktop computer in under 10 minutes.

In addition to the information given for the Linux corpus, we add several fields here particular to hierarchical corpora. The Monroe corpus has, in addition to the 10 top-level goal schemas, 38 subgoal schemas. The plans in the corpus were on average 3.8 subgoals deep. This measures how many nodes away each atomic action is from the top-level goal. The deepest atomic action in the corpus was 9 levels away from the top-level goal. Although the Monroe corpus provides hierarchical subgoal information, in this paper we only report on results of trying to predict the top-level goal (which we term *flat goal recognition*). We are currently using the Monroe corpus for training and testing a *hierarchical* goal recognizer.

3 Statistical Goal Recognition

We model goals, not as monolithic entities, but rather as goal *schemas* which are instantiated with parameters. As stated above, our goal is to do *instantiated* goal recognition, which is recognition of both the schema and its parameter values.

Towards this goal, we have previously built separate systems for doing stand-alone statistical goal *schema* recognition [Blaylock and Allen, 2003] and goal *parameter* recognition [Blaylock and Allen, 2004], and reported their separate performance on the Linux corpus.

The main contributions of the current paper are twofold: (1) we report the performance of the schema and parameter recognizers on a new corpus (Monroe), and (2) we describe our work on combining these recognizers to create a unified instantiated goal recognizer and report its performance on both the Linux and Monroe Corpora.

In the remainder of this section, we first present some preliminary mathematical definitions and then a mathematical formulation of the goal recognition problem. We then briefly describe our goal schema and parameter recognizers and report their performance on the Monroe corpus. We then de-

Goal Schema	English Description
`find-file-by-ext($extension)`	Find a file that ends in '.$extension'
`find-file-by-name($filename)`	Find a file named '$filename'
`know-filespace-usage-file($filename)`	Find out how much space file '$filename' uses
`know-filespace-free($partition_name)`	Find out how much filespace is used on filesystem '$partition_name'
`move-files-by-name($filename,$dirname)`	Move all files named '$filename' to a (preexisting) directory named '$dirname'
`move-files-by-size-lt($numbytes,$dirname)`	Move all files with less than $numbytes bytes to a (preexisting) directory named '$dirname'

Table 2: A few goal schemas from the Linux corpus

Goal Schema	English Description
`fix-power-line($location)`	Repair a power line at $location
`provide-medical-attention($person)`	Attend to a medical emergency involving $person
`provide-temp-heat($person)`	Provide temporary heating for $person
`quell-riot($location)`	Break up a riot occurring at $location
`set-up-shelter($location)`	Set up an emergency shelter at $location

Table 3: A few goal schemas from the Monroe corpus

scribe our instantiated goal recognizer and report its performance on both the Linux corpus and the Monroe corpus.

3.1 Preliminary Definitions

For a given domain, we define a set of goal schemas, each taking q parameters, and a set of action schemas, each taking r parameters. If actual goal and action schemas do not have the same number of parameters as the others, we can easily pad with 'dummy' parameters which always take the same value.[4]

Given an instantiated goal or action, it is convenient to refer to the schema of which it is an instance as well as each of its individual parameter values. We define a function `Schema` that, for any instantiated action or goal, returns the corresponding schema. As a shorthand, we use $X^S \equiv \texttt{Schema}(X)$, where X is an instantiated action or goal.

To refer to parameter values, we define a function `Param` which returns the value of the kth parameter value of an instantiated goal or action. As a shorthand we use $X^k \equiv \texttt{Param}(X, k)$, where X is again an instantiated action or goal.

As another shorthand, we refer to number sequences by their endpoints:

$$1, n \equiv 1, 2, \ldots, n$$

This allows us to shorten definitions in the following ways:

$$A_{1,n} \equiv A_1, A_2, \ldots, A_n$$

$$A_{1,n}^{1,r} \equiv A_1^1, A_1^2, \ldots, A_1^r, A_2^1, A_2^2, \ldots, A_{n-1}^r, A_n^1, \ldots, A_n^r$$

[4]The requirement that goal and action schemas have the same number of parameters is for convenience in the mathematical analysis. Below we report how this is circumstance is handled within the recognizer itself.

3.2 Problem Formulation

We define goal recognition as a classification task: given an observed sequence of n instantiated actions observed thus far ($A_{1,n}$), find the most likely instantiated goal g:

$$g = \text{argmax}_G \; P(G|A_{1,n}) \tag{1}$$

If we expand the goal and actions into their schemas and parameters, this becomes:[5]

$$g = \text{argmax}_{G^S, G^{1,q}} \; P(G^S, G^{1,q}|A_{1,n}^S, A_{1,n}^{1,r}) \tag{2}$$

We make two simplifying assumptions at this point, in order to make recognition more tractable. We briefly mention them here and then discuss them in more detail later on. First, we assume that goal parameters are independent of one another, and second, that goal schemas are independent from action parameters (given their action schemas). Given these assumptions, Equation 2 becomes:

$$g = \text{argmax} \; P(G^S|A_{1,n}^S) \prod_{j=1}^{q} P(G^j|G^S, A_{1,n}^S, A_{1,n}^{1,r}) \tag{3}$$

In Equation 3, the first term describes the probability of the goal schema G^S, which we use for goal schema recognition. The other terms describe the probability of each individual goal parameter G^j, which we estimate with our goal parameter recognizer.

Independence Assumptions

In formulating the problem above, we noted that we make two simplifying assumptions.

[5]From now on we drop the argmax subscript when context makes it obvious.

First, we make the simplifying assumption that all goal parameters are independent of one another. This allowed us to separate the probability of each parameter value into independent terms in Equation 3. This is, of course, not always the case — an obvious example from the Linux domain is that the source and destination parameters for a goal of copying a file should not have the same value. However, in many cases it appears that they are fairly independent.

We also assume that a goal schema is independent from an action's parameter values, given the action schema, which allows us to simplify the first term in Equation 3. This is also admittedly not always the case. In the Monroe domain, the `call` action describes a telephone call, with one parameter: the recipient of the call. This is used in the domain to turn off power to a particular location or to declare a curfew, as well as other things. The first use always has a power company as a parameter value whereas the second use includes a call to the local police chief.

Although conditioning on parameter values could be informative, it is likely that it would introduce sparsity problems because of the large number of possible parameter values.

We should also note that implicit in our definitions is the assumption that the agent only has a single goal which it is pursuing. As we discuss below, we are currently working towards extending our recognizer to perform *hierarchical* goal recognition, i.e., the recognition of all active subgoals in a hierarchical plan. This should allow us to handle the case where an agent pursues several goals serially. It is unclear, however, how our recognizer could be extended to handle the general case where an agent *simultaneously* pursues more than one goal.

3.3 Goal Schema Recognition

In previous work, we built a goal schema recognizer based on a bigram approximation of the first term from Equation 3 [Blaylock and Allen, 2003]:

$$g^S = \text{argmax}\ P(G^S) \prod_{i=2}^{n} P(A_i^S | A_{i-1}^S, G^S) \qquad (4)$$

The recognizer decides whether to make a prediction based on a confidence threshold τ. If the probability of the prediction is greater than τ, the recognizer makes a prediction; otherwise, it doesn't. The recognizer can also make n-best predictions, instead of just a single best prediction. This can be useful, as many applications which use goal recognition can do further domain-specific processing on the n-best list to make a better prediction.

The complexity of the schema recognizer is $O(|G|)$, where G is the set of goal schemas in the domain.

Schema Recognition Experiments

We have elsewhere reported the results of training and testing a bigram-based goal schema recognizer on the Linux corpus [Blaylock and Allen, 2004] (repeated here for comparison). Using the same methods, we trained and tested the bigram schema recognizer on the Monroe corpus. In doing so, we randomly chose 4500 entries as training data, and tested on the remaining 500. The recognition results for the Linux corpus and Monroe corpus are shown in Table 4.[6]

We report results with metrics that are designed to measure the general goal recognition requirements described above. *Precision* and *recall* report the number of correct predictions divided by total predictions and total prediction opportunities, respectively. *Convergence* and *convergence point* stem from the fact that, oftentimes, the recognizer will be unsure very early on in a session, but may at some point 'converge' on the correct answer, predicting it from that point on until the end of the plan session. *Convergence* measures the percentage of plan sessions where the correct answer was converged upon.[7] For those plan sessions which converge, *convergence point* reports the average action observation after which it converged divided by the average number of actions for the converged sessions. This is an attempt to measure how *early* in the plan session the recognizer was able to zero in on the correct answer.

Results on the Monroe corpus were exceptionally better than those of the Linux corpus. This is likely due to several factors. First, the Monroe corpus is much larger in terms of training data, yet smaller in terms of possible goal schemas. Also, as the Monroe corpus was automatically generated, it does not include the kind of noisy data present in the Linux corpus (see discussion above). In addition, we believe this can partly be attributed to the fact that the Linux domain is difficult to map onto a plan representation (cf. discussion in [Lesh, 1998]). For example, many Linux commands (e.g., `ls`) can take an (almost arbitrary) set of flags (e.g., `-a`, `-l`, `-x`), each of which changes the results (and thus the effects in a plan representation). For example, a file's size is visible using `ls` with the `-l` flag, but not without.

Although the results for the Linux corpus aren't dismal (note the jump in precision for 2-best prediction), we believe good results in the Monroe domain indicate that the method may be more successful in domains which are more naturally modelled by planning operators.

In the Monroe domain, we get a precision of 95.6% for 1-best prediction, which can be raised to 99.4% by predicting the 2 best schemas. In 2-best prediction, the correct schema is eventually predicted for 99.8% of sessions. For 1-best, the recognizer converges on the correct schema after seeing an average of 5.4 of 10.2 actions (for those cases which converge). This means that, on average, the recognizer is sure about the prediction a little more than halfway through the session.

Recall for 1-best is 55.2%, which increases to 69.6% for 3-best prediction. Although this may seem poor in comparison to precision and convergence numbers in the 90's, it is important to consider that we are doing goal recognition. A recall of 100% would mean that the algorithm made the correct prediction directly after the first action in each session and converged on that prediction at that point. In all but the

[6] The threshold value τ needs to be individually set for each n-best value. The τ values used here were chosen experimentally: i.e., by trying several and picking the best one.

[7] This essentially measures how many *last* predictions were correct, i.e., whether we *ended* predicting the right answer.

	Linux			Monroe		
N-best (τ)	**1** (0.4)	**2** (0.6)	**3** (0.9)	**1** (0.7)	**2** (0.9)	**3** (0.9)
Precision	37.6%	64.1%	73.7%	95.6%	99.4%	98.9%
Recall	22.9%	40.6%	41.4%	55.2%	58.7%	69.6%
Convergence	37.4%	56.5%	59.7%	96.4%	99.8%	100.0%
Convergence Point	3.5/5.9	4.0/7.2	4.1/7.2	5.4/10.2	5.4/10.3	4.1/10.2

Table 4: Goal schema recognition results

simplest domains, such a feat is not likely possible (even for humans), given the inherent ambiguity in most plan recognition domains. (As an example, many Linux sessions began with the `pwd` command, which gives little indication of what the user's goal is.) We believe 55.2% recall (or 69.6% for 3-best) to be a good result.[8]

3.4 Goal Parameter Recognition

We have also introduced a goal *parameter* recognizer loosely based on the second term of Equation 3 [Blaylock and Allen, 2004]. Input to the parameter recognizer is the action sequence *and* the (known) goal schema for the session, as the parameter recognizer expects to *know* the correct goal schema a priori. Additionally, the parameter recognizer works on each parameter position in the goal schema independently. We deal with these constraints below in building our integrated instantiated goal recognizer.

The parameter recognizer uses a tractable subset of Dempster-Schafer Theory (DST) to estimate the probability that action parameters seen so far are the values of goal parameters. It also uses the measure of ignorance from DST (Ω) to decide if it is confident enough to make a prediction. If the probability of the prediction is greater than Ω, the recognizer predicts.

As we discuss in more detail below, precision in parameter recognition turns out to be a big factor in the overall performance of our instantiated goal recognizer. In order to increase precision, we add an additional factor to our previous parameter recognizer which we call *ignorance weight* (ψ). In deciding whether or not to make a prediction, Ω is multiplied by ψ before it is compared with the probability of the prediction. Higher values of ψ will cause the recognizer only to predict when more sure of the prediction. Using this factor, we were able to use this to raise the precision of the parameter recognizer considerably, as reported below.

For a single parameter prediction, the time complexity of the parameter recognizer is $O(i)$ where i is the number of actions observed so far. Note that this is scalable, as it is neither dependent on the number of goal schemas in the domain *nor* the number of potential values for the parameter (e.g., objects in the domain).

[8]It is been difficult to compare plan recognizers, given the historical lack of data and even common metrics for reporting results. We hope that the introduction of plan corpora as test sets and standard performance measurements will allow closer comparison in the future.

Parameter Recognition Experiments

Elsewhere, we have reported the results of the parameter parameter recognizer on the Linux corpus (repeated here for comparison) [Blaylock and Allen, 2004]. In addition, we tested different values of the ignorance factor (ψ) and show the results for $\psi = 2.0$. We also tested the parameter recognizer for different values of ψ on the Monroe corpus, randomly dividing it into training (4500 sessions) and testing (500 sessions) as done above with the schema recognizer. The results are shown in Table 5.

For parameter recognition, we use the same metrics used for reporting schema recognition. In addition, we use two additional metrics: *recall/feasible* and *convergence/feasible*. This stems from the fact that the parameter recognizer can *only* recognize objects as parameter values that it has seen in the observed actions in the session.[9] This means that if the goal parameter value is not used until the fifth action in the session, it is not feasible for the recognizer to correctly identify it in its first four predictions. In the Linux domain, only 56.1% of predictions are feasible for the recognizer, and in Monroe only 49.6%. *Recall/Feasible* is a measure of recall for those predictions which were feasible for the recognizer. *Convergence/Feasible* works similarly for convergence. In only 82.1% of sessions did the goal parameter value ever occur as an action parameter value, and for Monroe in 79.4% of sessions.

As is shown, precision for both Linux and Monroe was greatly improved by using the higher ψ value, although at a cost to recall (as would be expected). This is especially true for the 1-best case for the Monroe domain, where precision moves from 77.1% to 94.3%.

In comparing the domains in the $\psi = 2$ case, performance on the Monroe domain is slightly better than that for Linux, with, for example, 94.3% precision for the 1-best case in Monroe versus 90.9% in Linux. In this case, recall is lower in Monroe (27.8% versus 32.1%). Recall is less than 40% for both corpora for 2-best, although, as mentioned above, low recall is to be expected in general for goal recognition.

3.5 Instantiated Goal Recognition

We now turn our attention to building an *instantiated* goal recognizer using the schema and parameter recognizers. This question brings us back to our original formulation of goal recognition above, particularly to Equation 3. We have a goal

[9]This is because the parameter recognizer works on probabilities of parameter *relations* between goal and action parameters. The advantages and disadvantages of this approach are discussed in [Blaylock and Allen, 2004].

	Linux				Monroe			
	1-best		2-best		1-best		2-best	
Ignorance weight (ψ)	**1.0**	**2.0**	**1.0**	**2.0**	**1.0**	**2.0**	**1.0**	**2.0**
Precision	84.3%	90.9%	84.4%	93.2%	77.1%	94.3%	88.6%	97.6%
Recall	37.2%	32.1%	42.1%	35.8%	38.5%	27.8%	44.9%	39.2%
Recall/Feasible	66.3%	57.1%	75.0%	63.8%	77.5%	55.9%	90.5%	78.9%
Convergence	64.4%	54.4%	70.5%	60.3%	61.2%	46.9%	76.2%	76.2%
Conv./Feasible	78.4%	66.2%	85.9%	73.5%	77.1%	59.1%	96.1%	96.1%
Convergence Point	3.2/5.8	3.5/6.2	3.0/5.8	3.4/6.2	4.4/9.4	5.1/10.0	3.8/9.1	4.7/9.0

Table 5: Goal parameter recognition results on the Linux corpus and Monroe corpus

schema recognizer which estimates the first term, and a goal parameter recognizer which estimates the each of the terms for each parameter position in a goal schema. Mathematically, we simply need to compute the argmax to get the most likely instantiated goal, although, as we will see, this is not so straightforward, especially if we want to support n-best and partial prediction.

The argmax in Equation 3 above is an optimization problem over several variables ($G^S, G^{1,q}$), where q is the arity of the goal schema. Although this could mean a big search space, it remains tractable in the 1-best case because of an assumption made above: namely, that goal parameter values are independent of one another (given the goal schema). This means that, given a goal schema g^s, the set of individual argmax results for each goal parameter g^j is guaranteed to be the maximum for that goal schema. This now becomes an optimization problem over just two variables: the goal schema and its parameters.

Although this works well in theory, there are several problems with using it in practice. First, the argmax only gives us the 1-best prediction. The search space gets larger if we want the n-best predictions. Second is the problem mentioned earlier about goal schema arity. Straight probability comparisons will not work for goals with different arities, as lower-arity goals will tend to be favored.

Partial prediction is also a problem. We want to support partial predictions by allowing the recognizer to predict a (possible empty) subset of the parameter values for a goal schema. This allows us to make predictions even in cases where the parameter recognizer is unsure about a specific parameter, and capitalizes on the ability of the stand-alone parameter recognizer to not make a prediction in cases where it is not certain.

In doing partial predictions, however, we encounter a natural tension. On one hand, we want the predictions to be as specific as possible (e.g., predict as many parameter values as possible). On the other hand, we want high precision and recall for predictions.[10] The full-prediction recognizer gives us specific predictions (with all parameters predicted), but would likely have low precision/recall. At the other extreme, we could just predict the goal schema which would give us the best chance for high precision/recall. Another dilemma is how to compare two predictions when one has more predicted parameters than the other.

Because of these problems, we have decided to take a slightly different approach to building our instantiated goal recognizer which capitalizes on the prediction ability of the schema and parameter recognizers as they are.

Our instantiated goal recognizer works as follows: at each observed action, we first run the goal schema recognizer. This makes an n-best prediction of schemas (or doesn't if the confidence threshold isn't reached). If no prediction is made by the schema recognizer, the instantiated recognizer also makes no prediction. If the schema recognizer does make a prediction, we use the parameter recognizer to make (or not make) 1-best predictions for each of the parameter positions for each of the n-best schemas. This automatically gives us partial prediction if a prediction is not made for one or more parameter positions in a schema. The combined results then form the n-best instantiated prediction.

The complexity of the instantiated recognizer is $O(|G|iq)$ as the main loop runs the parameter recognizer ($O(i)$) for each parameter (q) for each goal schema ($|G|$).[11] If we assume that q is relatively small, the complexity becomes $O(|G|i)$, which is linear in the number of goal schemas and the number of actions observed so far.

A final item we must mention is that this algorithm does not give us true n-best results for the search space. It instead chooses the n-best goal schemas, and then (selectively) predicts parameters for them. A true n-best result would include the possibility of having a goal schema twice, with different predictions for parameters. However, as mentioned above, we did not see an obvious way of deciding between, for example, a goal schema with no parameters predicted, and that same goal schema with one parameter predicted. The latter is guaranteed to not have a lower probability, but it is a more specific prediction. Although we don't provide true n-best prediction, we believe our algorithm provides a natural way of deciding between such cases by appealing to the parameter recognizer itself.

Goal Recognition Experiments

We tested the instantiated recognizer on the Linux corpus and Monroe corpus in a similar way to the experiments described above. The results are shown in Table 6.

[10] In a way, specificity adds a third dimension to the existing tension between precision and recall.

[11] Note that, although we only need run the parameter recognizer on the n-best schemas to get immediate results, we still need to run it to keep the probability assignments for the other parameters up to date.

	Linux			Monroe		
N-best (τ/ψ)	**1** (0.4/2.0)	**2** (0.6/2.0)	**3** (0.9/2.0)	**1** (0.7/2.0)	**2** (0.9/2.0)	**3** (0.9/2.0)
Precision	36.2%	60.2%	68.8%	93.1%	94.6%	96.4%
Recall	22.1%	38.1%	38.7%	53.7%	56.6%	67.5%
ParamPctg	51.5%	50.0%	51.6%	20.6%	21.8%	22.3%
Convergence	36.1%	53.8%	56.5%	94.2%	97.4%	98.6%
ConvParamPctg	51.8%	49.0%	49.4%	40.6%	41.1%	48.4%
Convergence Point	3.6/5.8	4.0/7.0	4.1/7.0	5.4/10.0	5.5/10.1	4.4/10.2

Table 6: Instantiated goal recognition results

We report results with the same measures used for schema recognition. In addition, we use two new measures. *ParamPctg* reports, for all correct predictions, the percentage of the goal parameters that were predicted. *ConvParamPctg* reports the same for all sessions which converged. These are an attempt to measure the specificity of correct predictions which are made.

Results followed the performance of the schema recognizer quite closely, being of course slightly lower with the addition of parameter recognition, although the relationship was not linear. Interestingly enough, ParamPctg and ConvParamPctg stayed fairly constant over n-best values for each corpus. In the Linux corpus, correct predictions had around 50% of their parameters instantiated, while the Monroe corpus had only about 21%. Although both corpora had fairly comparable performance in (stand-alone) parameter recognition, it appears that a greater portion of the correctly predicted goals in the Monroe domain happened to be goals for which the parameter recognizer didn't have as high of recall.

Although space precludes us from reporting all experimental results, we should mention that we found it to be extremely important to have high precision from the parameter recognizer. As an example, in our experiments with the unchanged parameter recognizer (i.e., $\psi = 1.0$), precision for the 3-best case for the Monroe corpus was only 73.0% (compared to 96.4%) and recall was only 51.3% (compared to 67.5%). Of course, the lower threshold to parameter prediction boosted ParamPctg to 36.9% (from 22.3%).

Recognition times averaged about 0.4 seconds per action for Linux and 0.2 seconds per action for Monroe, running unoptimized Perl code on a high-end desktop PC.

4 Related Work

Goal and plan recognizers are typically divided into those based on logical consistency, and those that are probabilistic. Logical-consistency recognizers typically cannot distinguish among goals that are consistent with the observations, and usually do not have the predictive power needed for online recognition. We will, therefore, concentrate our comments on probabilistic goal recognizers.

There has been much work on probabilistic goal schema recognition, although very little experimental results have been reported (likely due to a lack of corpora in the field). [Bauer, 1995] uses Dempster-Shafer Theory to do goal schema recognition, but, as the recognizer uses full Dempster-Shafer Theory, it is not clear if the recognizer would be tractable in general. Several groups (e.g., [Charniak and Goldman, 1993; Huber *et al.*, 1994; Horvitz and Paek, 1999]) use Belief Networks (BNs) to do goal recognition. However, the size of these networks must either be large from the start, or they grow as the number of observed actions increases; reasoning with (large) BNs is intractable in general.

[Albrecht *et al.*, 1998] use a Dynamic Belief Network (DBN) to do goal schema recognition[12] in a way similar to our schema recognizer. However, the system did not perform *instantiated* recognition, and treated goals as atomic units. The recognizer also did not perform n-best, partial prediction, but rather returned a probability distribution over the goals.

[Pynadath and Wellman, 2000] and [Bui, 2003] use DBNs to cast (hierarchical) goal schema recognition as something akin to parsing. Both, however, are dependent on the ability of the system to be able to perceive or accurately estimate when higher-level goals (constituents) terminate, and it is unclear if the recognizers would be accurate in domains where this is difficult to predict. In addition, these systems also do not work on goal parameters.

Although parameter prediction has been included in logical-based (e.g., [Kautz, 1991]) and case-based (e.g., [Cox and Kerkez, to appear]) plan recognizers, relatively little attention has been given it in work on probabilistic plan recognizers. Probabilistic systems which do include this typically use probabilities for goal schema prediction, but logical-based methods for filling in parameters (e.g., [Bauer, 1995]). The recognizer in [Charniak and Goldman, 1993] dynamically constructs a Belief Network (BN) with nodes for action and goal schemas, objects (possible parameter values), and relations that use the latter to fill slots in the former. For a given parameter slot, however, they consider all objects of the correct type to be equally likely, whereas we distinguish these using probabilities learned from a corpus. As the network grows at least with the number of observed actions (and likely the number of goal schemas), it is unclear if this approach would be scalable in general.

5 Conclusions and Future Work

We have presented an instantiated goal recognizer based on machine learning which is fast, scalable, and able to make partial predictions. We reported the performance of the recognizer on two plan corpora.

[12] as well as next state and action schema prediction

We are currently extending the recognizer to perform *hierarchical* goal recognition, which we define to be the recognition of the chain of active subgoals up to the top goal (cf. [Bui, 2003]). Complex plans covering longer time-scales are less likely to be identifiable from a few observations alone (which tend to reflect more immediate subgoals). Ideally, we would want to recognize subgoals for partial results, even if it is not immediately clear what high-level goal is.

Acknowledgments

We would like to thank the anonymous reviewers for their suggestions and comments.

This material is based upon work supported by a grant from DARPA under grant number F30602-98-2-0133; two grants from the National Science Foundation under grant number IIS-0328811 and grant number E1A-0080124; and the EU-funded TALK project (IST-507802). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the above-mentioned organizations.

References

[Albrecht *et al.*, 1998] David W. Albrecht, Ingrid Zukerman, and Ann E. Nicholson. Bayesian models for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction*, 8:5–47, 1998.

[Allen *et al.*, 2001] James F. Allen, Donna K. Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent. Towards conversational human-computer interaction. *AI Magazine*, 22(4):27–37, 2001.

[Bauer and Paul, 1993] Mathias Bauer and Gabriele Paul. Logic-based plan recognition for intelligent help systems. In Christer Bäckström and Erik Sandewall, editors, *Current Trends in AI Planning: EWSP '93 — Second European Workshop on Planning*, Frontiers in Artificial Intelligence and Applications, pages 60–73. IOS Press, Vadstena, Sweden, December 1993. Also DFKI Research Report RR-93-43.

[Bauer, 1995] Mathias Bauer. A Dempster-Shafer approach to modeling agent preferences for plan recognition. *User Modeling and User-Adapted Interaction*, 5(3–4):317–348, 1995.

[Blaylock and Allen, 2003] Nate Blaylock and James Allen. Corpus-based, statistical goal recognition. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1303–1308, Acapulco, Mexico, August 9–15 2003.

[Blaylock and Allen, 2004] Nate Blaylock and James Allen. Statistical goal parameter recognition. In Shlomo Zilberstein, Jana Koehler, and Sven Koenig, editors, *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS'04)*, pages 297–304, Whistler, British Columbia, June 3–7 2004. AAAI Press.

[Blaylock and Allen, 2005] Nate Blaylock and James Allen. Generating artificial corpora for plan recognition. In *International Conference on User Modeling (UM'05)*, Edinburgh, July 2005. To appear.

[Bui, 2003] Hung H. Bui. A general model for online probabalistic plan recognition. In Georg Gottlob and Toby Walsh, editors, *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, August 9–15 2003.

[Carberry, 1990] Sandra Carberry. *Plan Recognition in Natural Language Dialogue*. MIT Press, 1990.

[Charniak and Goldman, 1993] Eugene Charniak and Robert P. Goldman. A Bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53–79, 1993.

[Cox and Kerkez, to appear] M. T. Cox and B. Kerkez. Case-based plan recognition with novel input. *International Journal of Control and Intelligent Systems*, to appear.

[Horvitz and Paek, 1999] Eric Horvitz and Tim Paek. A computational architecture for conversation. In *Proceedings of the Seventh International Conference on User Modeling*, pages 201–210, Banff, Canada, June 1999. Springer-Verlag.

[Huber *et al.*, 1994] Marcus J. Huber, Edmund H. Durfee, and Michael P. Wellman. The automated mapping of plans for plan recognition. In R. L. de Mantaras and D. Poole, editors, *UAI94 - Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, pages 344–351, Seattle, Washington, 1994. Morgan Kaufmann.

[Kautz, 1991] Henry Kautz. A formal theory of plan recognition and its implementation. In J. Allen, H. Kautz, R. Pelavin, and J. Tenenberg, editors, *Reasoning about Plans*, pages 69–125. Morgan Kaufman, San Mateo, CA, 1991.

[Lesh *et al.*, 1999] Neal Lesh, Charles Rich, and Candace L. Sidner. Using plan recognition in human-computer collaboration. In *Proceedings of the Seventh International Conference on User Modeling*, Banff, Canada, June 1999. Springer-Verlag. Also available as MERL Technical Report TR98-23.

[Lesh, 1998] Neal Lesh. *Scalable and Adaptive Goal Recognition*. PhD thesis, University of Washington, 1998.

[Pynadath and Wellman, 1995] David. V. Pynadath and Michael. P. Wellman. Accounting for context in plan recognition, with application to traffic monitoring. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 472–481, Montreal, Canada, 1995. Morgan Kaufmann.

[Pynadath and Wellman, 2000] David V. Pynadath and Michael P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-2000)*, pages 507–514, Stanford, CA, June 2000.

[Stent, 2000] Amanda J. Stent. The Monroe corpus. Technical Report 728, University of Rochester, Department of Computer Science, March 2000.