

Statistical Goal Parameter Recognition*

Nate Blaylock and James Allen

Department of Computer Science
University of Rochester
Rochester, New York, USA
{blaylock,james}@cs.rochester.edu

Abstract

We present components of a system which uses statistical, corpus-based machine learning techniques to perform *instantiated goal recognition* — recognition of both a goal schema and its parameter values. We first present new results for our previously reported statistical goal schema recognizer. We then present our goal parameter recognizer. The recognizer is fast (linear in the number of observed actions) and is able to recognize parameter values which were never seen in training. In addition, this is the first system we are aware of that uses statistical methods to recognize goal parameter values.

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 (Lesh, Rich, & Sidner 1999), traffic monitoring (Pynadath & Wellman 1995), and dialogue systems (Carberry 1990; Allen *et al.* 2000).

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

1. **Speed:** Most applications use goal recognition “online” and must 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 most 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 apply supervised machine-learning techniques to the tasks of goal schema recognition and goal parameter recognition. This has two advantages (which correspond to the two desired traits of goal recognizers described above). First, recognition is fast. The schema recognizer runs in time linear to the number of possible goal schemas, while the parameter recognizer runs in time linear to the number of observed actions. In contrast, most previous work (e.g., (Kautz 1991; Charniak & Goldman 1993)) has had runtimes that are exponential in the number of goals.¹ Subsequent work (e.g., (Vilain 1990; Albrecht, Zukerman, & Nicholson 1998; Pynadath & Wellman 2000)) has improved runtimes, but only at the expense of expressiveness — typically either totally-ordered plans, or the recognition of only goal schemas, but not their parameter values.

The use of statistics from a corpus allows the recognizer to be more predictive. Most logic-based recognizers (e.g., (Kautz 1991; Lesh 1998)) are sound (assuming a complete plan library), but are unable to distinguish between (often numerous) logically-consistent goals.

In the remaining sections, we first describe the plan corpus we use in our experiments. We then present the formulation for statistical instantiated goal recognition and discuss our previous work on goal schema recognition. We then present our model of goal parameter recognition. Finally, we briefly mention our current work on using the two recognizers together as a full goal recognizer. We then conclude and mention future work.

*This material is based upon work supported by a grant from the Department of Education under grant number P200A000306-02; a grant from DARPA/Air Force under grant number F30602-98-2-0133; a grant from Department of Defense/ONR under grant number N00014-01-1015; and a grant from the National Science Foundation under grant number E1A-0080124. 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.

¹In this paper, we will only refer to goal recognition, even though most previous work has tackled the harder problem of plan recognition (which is really recognition of an agent's goal *and* plan). In this sense, it is somewhat unfair to compare our goal recognizer to full-fledged plan recognizers. However, very little work has been done on goal recognition in its own right, so plan recognizers are all we have to compare against.

Plan Sessions	457
Goal Schemas	19
Action Schemas	43
Ave. Actions/Session	6.1

Table 1: The Linux corpus

The Linux Plan Corpus

In our preliminary work on goal schema recognition (Blaylock & Allen 2003), we used Lesh’s Unix plan corpus (Lesh 1998), which was gathered from human Unix users (CS undergraduates) at the University of Washington. Users were given a task in Unix (a goal), and were instructed to solve it using any Unix commands (with a few rules such as no pipes, no awk, etc.) The users’ commands and results were recorded, as well as whether or not they successfully accomplished the goal. There were 59 successful goal sessions which involved 11 different goals.

In order to test our work on a bigger data set, both in terms of more training data as well as more goal schemas, we gathered the Linux corpus. The Linux corpus was gathered from human Linux users (graduate and undergraduate students, faculty, and staff) from the University of Rochester Department of Computer Science in a manner similar to the Unix corpus. We were able to gather 457 successful goal sessions involving 19 different goal schemas. Other relevant data about the corpus is shown in Table 1. 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.

We use this corpus to train and test statistical models of goal recognition.

Statistical Goal Recognition

In most domains, goals are not monolithic entities; rather, they are *goal schemas* with instantiated parameters. In (Blaylock & Allen 2003), we describe a statistical goal schema recognizer. We briefly review that work here and report our most recent results. This will then serve as the context and motivation for our work on goal parameter recognition. Combining goal schema recognition with goal parameter recognition allows us to do *instantiated goal recognition*, which we will refer to here as simply *goal recognition*.

We first give some definitions which will be useful throughout the paper. We then describe the recognition model and our most recent results.

Definitions

For a 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.²

²The requirement that goal and action schemas have the same number of parameters is for convenience in the mathematical analysis and is not a requirement of the algorithm itself.

Given an instantiated goal or schema, 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 $X^S \equiv \text{Schema}(X)$, where X is an instantiated action or goal.

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

As another shorthand, we refer to number sequences by their endpoints: $1, n \equiv 1, 2, \dots, n$. Thus $A_{1,n} \equiv A_1, A_2, \dots, A_n$ and $A_{1,n}^{1,r} \equiv A_1^1, A_1^2, \dots, A_1^r, A_2^1, A_2^2, \dots, A_{n-1}^r, A_n^1, \dots, A_n^r$.

Problem Formulation

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

$$g = \operatorname{argmax}_G P(G|A_{1,n}) \quad (1)$$

If we expand the goal and actions into their schemas and parameters, this becomes:³

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

If we assume that goal parameters are independent of each other and that goal schemas are independent from action parameters (given their action schemas), this becomes:

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

In Equation 3, the first term describes the probability of the goal schema G^S , which we will use for goal schema recognition. The other terms describe the probability of each goal parameter G^j , which we discuss in detail later in the paper.

Goal Schema Recognition⁴

We use the first term from Equation 3 to recognize a goal schema g^S . By making a bigram independence assumption over observed actions, we get the following:

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

In training, we use the plan corpus to estimate prior goal schema probabilities ($P(G^S)$) and the bigram probabilities over observed action schemas ($P(A_i^S|A_{i-1}^S, G^S)$).

³From now on we drop the argmax subscript when the context makes it obvious.

⁴This section summarizes our previous work and presents some new results. For more details, please see (Blaylock & Allen 2003).

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

Table 2: A few goal schemas from the Linux corpus

During recognition, the recognizer first initializes its goal schema probabilities with $P(G^S)$. Then, for each action observed, it multiplies each goal's score by the corresponding conditional bigram probability and makes a prediction.

This algorithm has the nice feature of compositionality. For each goal, its probability after observation $i + 1$ is simply the new bigram probability times the overall probability after observation i . Because of this the complexity of making a prediction after each observation is $O(|G|)$, where G is the set of goal schemas in the domain.

Making Predictions Instead of letting the recognizer make a prediction after each observed action, we set a confidence threshold τ , which allows the recognizer to decide whether or not it is confident enough to make a prediction. If the probability of the prediction is greater than τ , the recognizer predicts. Otherwise, it predicts “don't know.”

For some applications where the result of goal recognition is used for further reasoning (e.g., natural language understanding for dialogue systems, which is where we plan to use the goal recognizer), we do not necessarily need a single prediction, but instead can use an n -best prediction as a measure of uncertainty. In the case of an n -best prediction, the probability of the prediction is taken to be the sum of the probabilities of the n individual goals, and that is then compared against τ in deciding whether to make a prediction.

Experiments

We measure several things for our experiments, based on our 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 observation, 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.⁵ For those plan sessions which converge, *convergence point* reports the average action observation after which it converged over the average number of actions for the converged sessions. This is an attempt to measure how *early* in the plan

⁵This basically measures how many *last* predictions were correct, i.e., whether we *ended* predicting the right answer.

N-best (τ)	1 (0.3)	2 (0.7)	3 (0.9)	4 (0.9)
Precision	35.1%	66.7%	73.7%	76.4%
Recall	23.6%	39.2%	41.4%	45.4%
Convergence	37.9%	55.1%	59.7%	64.6%
Convergence Point	3.5/5.9	4.1/7.2	4.1/7.2	4.0/7.2

Table 3: Goal schema recognition results on the Linux corpus using a bigram model

session the recognizer was able to zero in on the correct answer.

Because of the small size of the Linux corpus, we ran experiments using cross-validation training and testing over the whole corpus, testing on sets of 5 plan sessions at a time. Table 3 shows some of the results for different n -best values.⁶ For example, the recognizer was able to achieve 73.7% precision with 41.4% recall for the 3-best case (with 59.7% of the plan sessions converging). Note that although the average convergence point increases from 3.5 to 4.1 from 1-best to 2-best, the 2-best is actually converging earlier in the session percentage-wise, since the average total actions per converged session goes up from 5.9 to 7.2.

Discussion

At a first glance, the 1-best results are not very encouraging, and are much worse than our pilot results on the smaller Unix corpus (Blaylock & Allen 2003). Precision is only 35.1% and recall only 23.6%, with only 37.9% of sessions converging. However, things may not be as bad as they may seem from those numbers.

First of all, it appears that Linux is a hard domain. Wrong and missed predictions seem to come from several categories of problems. First, the data is from real humans and therefore noisy. Although we do some automatic filtering of the corpus before training, typos still exist (e.g., using the command `ln` instead of `ls`). More difficult to weed out are bad use of commands. For example, one user used the command `fgrep` to try to search for a file in a directory tree (`fgrep` is for searching files for text). Such mistakes often ruined an entire session.

⁶The threshold value τ needs to be individually set for each n -best value. The τ values chosen here were chosen experimentally.

In addition to noisy data, some of the goal schemas we chose for Linux are very similar (e.g., `find-file-by-ext` and `find-file-by-name`). The recognizer often confused these (and other similar sets of goals).

Also, although we represent the domain with “flat” plans, a natural subgoal relationship exists between some of the goal schemas, which also confused the recognizer. For example, in order to perform a `move-files-by-name`, a user has to first find the files (`find-file-by-name`) and the directory to move them to (`find-dir-by-name`). The recognizer would often predict one of the “subgoals” instead of the real goal schema.

Taking all of that into account, let’s look at the results one more time. Precision and convergence increase substantially if we take the 2-best results. This seems, in a large part, to be due to the fact that similar goal schemas could now be predicted together. A precision of 66.7% is actually not too bad.

Recall, however, remains fairly low: only 39.2% for the 2-best case. It is important to note that, in all but the most trivial domains, 100% recall will probably be impossible, even for humans. In order to have 100% recall, we would have to be able to predict, and converge on, the correct goal after the agent’s very first action. In the Linux domain, for example, a first action of `pwd`, which has very little predictive power, was not uncommon. Also, for goals like `move-files-by-name` it is not always clear that the goal is a move (as opposed to a copy, for example) until the very last action (which is typically the `mv` command).

We believe that this system will do better on other domains. We are currently testing it in a more “typical” domain of 911 emergencies. Also, we believe that the solution to some of these problems will be to move to a hierarchical decomposition model, which would allow the separate recognition of subgoals and end goals. We are currently working to extend the model to handle hierarchical plans.

Goal Parameter Recognition

The above system only recognizes goal schemas. To do full goal recognition, we also need to recognize each parameter value for the goal schema.

One straightforward way of doing this would be to treat instantiated goal schemas as atomic goals and then use the goal schema recognition algorithm from above. Thus, instead of estimating $P(\text{move-files-by-name}|ls,cd)$, we would estimate $P(\text{move-files-by-name}(a.txt,bdir)|ls(papers),cd(progs))$.

This solution has several problems. First, this would result in an exponential increase in the number of goals, as we would have to consider all possible ground instances. This would seriously impact the speed of the algorithm. It would also affect data sparseness, as the likelihood to have seen any n-gram in the training data will decrease substantially.

Instead, we perform goal schema and parameter recognition separately, as described in Equation 3 above. From the last term of the equation, we get the following for a single parameter g^j :

$$g^j = \operatorname{argmax} P(G^j|G^S, A_{1,n}^S, A_{1,n}^{1,r}) \quad (5)$$

We could estimate this with an n-gram assumption as we did above. However, there are several problems here as well. First, this would make updates at least linear in the number of objects in the world (the domain of g^j), which may be expensive in domains with many objects. Second, even without a large object space, we may run into data sparsity problems, since we are including both the action schemas and their parameter values.

The solution above misses out on the generalization that, oftentimes, the *positions* of parameters are more important than their value. For example, the first parameter (i.e., the *source file*) of the action `mv` is often the `$filename` parameter of the goal `move-files-by-name`, whereas the second parameter (i.e., the *destination*) almost never is, regardless of the parameter’s actual value. Our model learns probability distributions of equality over goal and action parameter positions. During recognition, we use these distributions along with a special, tractable case of Dempster-Shafer Theory to dynamically create a set of possible parameter values and our confidence of them, which we use to estimate Equation 5.

In this section we first describe this model and then report on experiments using it on the Linux corpus.

Our Model

Formally, we want to learn the following probability distribution: $P((G^j = A_i^k)|G^S, A_i^S)$. This gives us the probability that the value of the k th parameter of action A_i is equal to the j th parameter of the goal G , given both the goal and action schemas as well as the two parameter positions. Note that the *value* of the parameter is not considered here, only the *position*. We can easily compute this conditional probability distribution from our training corpus.

To use the above model to predict the value of each goal schema parameter as we make action observations, we need to be able to combine probabilities for each parameter in the observed action, as well as probabilities from action to action. For this we use Dempster-Shafer Theory.

Dempster-Shafer Theory Dempster-Shafer Theory (DST)⁷ is a generalization of probability theory which allows for incomplete knowledge. Given a domain Ω , a probability mass is assigned to each subset of Ω , as opposed to each element, as in classical probability theory. Such an assignment is called a *basic probability assignment* (bpa).

Assigning a probability mass to a subset in a bpa means that we place that level of confidence in the subset, but cannot be any more specific. For example, suppose we are considering the outcome of a die roll ($\Omega = \{1, 2, 3, 4, 5, 6\}$).⁸ If we have no information, we have a bpa of $m(\Omega = 1)$, i.e., all our probability mass is on Ω . This is because, although we have no information, we are 100% certain that *one* of the elements in Ω is the right answer; we just can’t be more specific.

⁷See (Bauer 1995) for a good introduction.

⁸This example is taken from (Bauer 1995).

Now suppose we are told that the answer is an even number. In this case, our bpa would be $m(\{2, 4, 6\}) = 1$; we have more information, but we still can't distinguish between the even numbers. A bpa of $m(\{2, 4, 6\}) = 0.5$ and $m(\{1\}) = 0.5$ would intuitively mean that there is a 50% chance that the number is even, and a 50% chance that it is 1. The subsets of Ω that are assigned non-zero probability mass are called the *focal elements* of the bpa.

A problem with DST is that the number of possible focal elements of Ω is the number of its subsets, or $2^{|\Omega|}$. This is both a problem for storage as well as for time for bpa combination (as discussed below). For parameter recognition, we use a special case of DST which only allows focal elements to be singleton sets or Ω (the entire set). This, of course, means that the maximum number of focal elements is $O(|\Omega|)$. As we show below, this significantly decreases the complexity of bpa combination, allowing us to run in time linear with the number of actions observed so far, regardless of the number of objects in the domain.

Evidence Combination Two bpas m and m' representing different evidence can be combined into a new bpa using Dempster's rule of combination:

$$(m \oplus m')(A) = \frac{\sum_{B \cap B' = A} m(B)m'(B')}{\sum_{B \cap B' \neq \emptyset} m(B)m'(B')} \quad (6)$$

The complexity of computing this is $O(l_m l_{m'} |\Omega|)$, where l_m and $l_{m'}$ are the number of focal elements in m and m' respectively. Basically, the algorithm does a set intersection (hence the $|\Omega|$ term) for combinations of focal elements.⁹

As mentioned above, the complexity would be prohibitive if we allowed any subset of Ω to be a focal element. However, our special case of DST limits l_m and $l_{m'}$ to $|\Omega| + 1$. Also, because we only deal with singleton sets and Ω , we can do set intersection in constant time. Thus, for our special case of DST, the complexity of Dempster's rule of combination becomes $O(l_m l_{m'})$ or $O(|\Omega|^2)$ in the worst case.

As a final note, it is easily shown that our special case of DST is closed under Dempster's rule of combination. We omit a formal proof, but it basically follows from the fact that the set intersection of any two arbitrary focal-element subsets from special-case bpas can only result in Ω , a singleton set, or the empty set. Thus, as we continue to combine evidence, we are guaranteed to still have a special-case bpa.

Representing the Model with DST As stated above, we estimate $P((G^j = A_i^k) | G^S, A_i^S)$ from the corpus. For a given goal schema G^S and the i th action schema A_i^S , we define a *local bpa* $m_{i,k}^j$ for each goal and action parameter positions j and k s.t. $m_{i,k}^j(\{A_i^k\}) = P((G^j = A_i^k) | G^S, A_i^S)$ and $m_{i,k}^j(\Omega) = P((G^j \neq A_i^k) | G^S, A_i^S)$. This local bpa intuitively describes the evidence of a single goal parameter

⁹We only need consider the focal elements here, since non-focal elements have a probability mass 0, which will always make $(m \oplus m')(A) = 0$.

value from looking at just one parameter position in just one observed action. The bpa has two focal elements: $\{A_i^k\}$, which is a singleton set of the actual action parameter value, and Ω . The probability mass of the singleton set describes our confidence that that value¹⁰ is the goal parameter value. The probability mass of Ω expresses our ignorance, as it did in the die roll example above.¹¹

In order to smooth the distribution, we always make sure that both elements (Ω and A_i^k) are given at least a little probability mass. If either one is 1, a very small value is taken from that and given to the other.

There are several things worth noting here. First, we assume here that we already know the goal schema G^S . This basically means we have a two-step process where we first recognize the goal schema and then use that to recognize the parameters. Below, we discuss how we can combine these processes in a more intelligent way.

Secondly, if a goal schema has more than one parameter, we keep track of these and make predictions about them separately. As we discuss below, it is possible that we will be more confident about one parameter and predict it, whereas we may not predict another parameter for the same goal schema. This allows us to make more fine-grained partial predictions.

Lastly, we do not need to represent, enumerate or even *know* the elements of Ω . This means that we can handle domains where the set of possible values is very large, or in which values can be created or destroyed. (Both of these are properties of the Linux domain.)

Combining evidence As mentioned above, we maintain a separate *prediction bpa* m^j for each goal parameter position j . Each of these are initialized as $m^j(\Omega) = 1$, which indicates complete ignorance about the parameter values.

As we observe actions, we combine evidence within a single action and then among single actions. First, within a single action i , we combine each of the local bpas $m_{i,k}^j$ for each parameter position k , which gives us an *action bpa* m_i^j . This describes the evidence the entire action has given us. Then, we combine the evidence from each observed action to give us an overall *prediction bpa* that describes our confidence in goal parameter values given all observed actions so far. We then use this prediction bpa to make (or not make) predictions. When we observe an action $A_i(p_1, p_2, \dots, p_r)$ we create local bpas for each action parameter position $m_{i,1}^j \dots m_{i,r}^j$. The action bpa m_i^j is the combination of all of these: $m_i^j = m_{i,1}^j \oplus m_{i,2}^j \oplus \dots \oplus m_{i,r}^j$.

¹⁰Note that this is the actual instantiated value and not just the position. Two instances of the same action schema with different parameter values will create different bpas.

¹¹Note here that Ω is the set of *all* possible domain values and still includes A_i^k . The reason for this is that just because we may not have seen much evidence for A_i^k given the action schema doesn't necessarily mean that A_i^k is *not* the goal parameter value. It just means that we don't yet have much evidence that *it is* the value. We actually ran experiments in which Ω did not include any of the values in the singleton focal elements and, while precision went up, recall dropped significantly.

The prediction bpa is similarly calculated from all of the action bpas from observed actions: $m^j = m_1^j \oplus m_2^j \oplus \dots \oplus m_i^j$. However, we can calculate this incrementally by calculating $m'^j = m^j \oplus m_i^j$ at each action observation. This allows us to only do 1 action-bpa combination per observed action.

It is worth noting here that only values that we have seen as an action parameter value will be part of the prediction bpa. Thus, the maximum number of focal elements for a bpa m^j will be the total number of unique action parameters seen, plus one (for Ω). As a corollary, this means that our method will not be able to correctly predict a goal parameter unless its value has been seen as an action parameter value in the current plan session. In the reported results below, we report results of total recall and also 'recall/feasible', which restricts recall to the prediction points at which the algorithm *had access to* the right answer. Admittedly, there are domains in which the correct parameter value could be learned directly from the training corpus, without having been seen in the current session, although we do not believe it will be so in most cases. We plan to investigate ways of learning both parameter values and positions in future work.

Prediction At some level, we are using the prediction bpa as an estimation of $P(G^j|G^S, A_{1,n}^S, A_{1,n}^{1,r})$ from Equation 5 above. However, because the bpa contains Ω , it is not a true probability distribution and cannot provide a direct estimation. Instead, we use Ω as a measure of confidence in deciding whether to make a prediction.

To make an n-best prediction, we take the n singleton sets with the highest probability mass and compare their combined mass with that of Ω . If their mass is greater, we make that prediction. If Ω has a greater mass, we are still too ignorant about the parameter value and hence make no prediction.

Complexity The other thing to mention is computational complexity of updating the prediction bpa for a single goal parameter G^j . We first describe the complexity of computing the i th action bpa m_i^j , and then the complexity of combining it with the previous prediction bpa m^j .

To compute m_i^j , we combine r 2-focal-element local bpas, one for each action parameter position. If we do a serial combination of the local bpas (i.e., $m_i^j = ((m_{i,1}^j \oplus m_{i,2}^j) \oplus m_{i,3}^j) \oplus \dots \oplus m_{i,r}^j$), this results in $r - 1$ combinations, where the first bpa is an intermediate composite bpa \hat{m}_i^j and the second is always a 2-element local bpa. Each combination (maximally) adds just 1 subset to \hat{m}_i^j (the other subset is Ω which is always shared). The $(k-1)$ th combination result $\hat{m}_{i,k-1}^j$ will have maximum length $k + 1$. The combination of that with a local bpa is $O(2(k + 1))$. Thus, the overall complexity of the combination of the action bpa is $\sum_{k=1}^{r-1} O(2(k + 1)) \approx O(r^2)$.

The action bpa m_i^j is then combined with the previous prediction bpa, which has a maximum size of $r(i - 1) + 1$

Nbest	1	2	3
Precision	84.3%	84.4%	84.7%
Recall	37.2%	42.1%	44.1%
Recall/Feasible	66.3%	75.0%	78.5%
Convergence	64.4%	70.5%	72.1%
Conv./Feasible	78.4%	85.9%	87.8%
Conv. Point	3.2/5.8	3.0/5.8	3.2/6.1

Table 4: Goal parameter recognition results

(from the number of possible unique action parameter values seen). The combination of the two bpas is $O(ir^2)$, which, together with the complexity of the computation of the action bpa becomes $O(ir^2 + r^2) \approx O(ir^2)$. r is actually constant here (and should be reasonably small), so we can treat it as a constant, in which case we get a complexity of $O(i)$. This is done for each of the q goal parameters, but q is also constant, so we still have $O(i)$. This gives us a fast parameter recognition algorithm which is not dependent on the number of objects in the domain.

Experiments

Results We tested our parameter recognizer on the Linux corpus by doing cross-validation training and testing similar to that described above for the schema recognizer. The results are shown in Table 4 for various n-best values. The metrics shown here mean the same as they did in the goal schema recognition results. We also report two new metrics: *recall/feasible* and *convergence/feasible*, both of which measure how much recall/convergence the recognizer got from what it could *feasibly* get. As mentioned above, the recognizer can only predict values that it has seen as action parameter values within the current session. Feasible recall for the corpus is 56.1%, meaning that at only 56.1% of the places where a prediction could have been made, the correct value had actually shown up in the action parameters. In fact, only in 82.1% of the goal sessions does the goal parameter value appear at all. Given these facts, the parameter recognizer actually does very well for the cases it can handle, both in terms of precision (84.7% for 3-best) and recall (44.1% for 3-best — 78.5% of feasible).

It is also interesting to note that, along those same measures, the feasible convergence point is 2.8/6.2, i.e., on average, for the sessions in which the correct parameter value does appear, it appears for the first time after 2.8 actions have been observed. By that measure, the convergence points of 3.2 and 3.0 mean that, in cases where the parameter prediction will converge, it usually converges right after it sees the correct value for the first time.

Discussion

As we note above, the parameter recognition results are good, especially compared to what is feasible for the algorithm. (Again, this appears to be a hard domain.) Some errors resulted from typographical errors from the user (e.g., typing `ls flie` instead of `ls file` and the recognizer predicting `flie`). More frequently, errors came from the

recognizer over-zealously predicting when it had not yet seen the correct value.

Much of missed feasible recall also appeared to be 'corpus noise', where the user used a command incorrectly. For example, for the goal `create-dir(s2,linux)` where the user was instructed to create a directory named `s2` inside a preexisting directory named `linux`, the users first command was `fgrep linux`, in an apparent attempt to find a directory named `linux`. While this made the correct value 'linux' feasible, it was assigned a very low probability mass, as `fgrep` searches for values in files, which is unrelated to this goal. The user went on to do a very long 29-command session using `cd` and `ls` to find the directory instead, and the recognizer mispredicted on all of them.

Full Goal Recognition

We have now presented a goal schema recognizer and a goal parameter recognizer. In this section we give a brief description of our current work of putting them together to form a complete goal recognizer. To perform full goal recognition, we combine both the schema and the parameter recognizers in finding the argmax described in Equation 4 above. Although the argmax is over several variables ($G^S, G^{1,q}$), computing it is tractable because of our assumption that goal parameter values are independent of each other. Given a goal schema g^S , the set of individual argmax results for each goal parameter g^j (described in Equation 5) is guaranteed to be the maximum for that goal.

To fully search for the argmax, we first perform goal schema recognition, and then do 1-best parameter recognition for each of the possible goal schemas. This has an overall complexity of $O(|G|i)$, which is still quite tractable. This gives us the most likely parameter instantiation for each of the goal schemas. We then choose the n-best of these by multiplying the posterior goal schema probability with the combined parameter probabilities.

Related Work

As mentioned above, most previous work on goal recognition has been very slow. In addition, non-probabilistic logic-based recognizers are typically not predictive enough for most domains, as they cannot choose among logically-consistent hypotheses. For this reason, we only discuss probabilistic recognizers here.

There has been much work on probabilistic goal schema recognition, although very little experimental results have been reported. (Bauer 1995) uses DST to do goal schema recognition, but, as it uses full DST, it is not clear if it is a tractable solution in general. (Charniak & Goldman 1993), (Huber, Durfee, & Wellman 1994), and (Horvitz & 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 will grow as the number of observed actions increases, and reasoning with BNs is intractable in general.

(Pynadath & Wellman 2000) and (Bui 2003) use Dynamic Belief Networks (DBNs) to cast plan recognition as something akin to parsing. Both, however, are dependent on the

ability of the system to be able to perceive (estimate) when higher-level plans terminate, and may not be suitable to the many domains where this is difficult to predict.

Probably the closest work to our goal schema recognizer is (Albrecht, Zukerman, & Nicholson 1998), which uses a dynamic belief network to do goal recognition in a multi-user dungeon domain. They use a bigram model of action schemas to predict goal schemas. Although our approaches are similar, there are a few significant differences. They encode state into their model (in the form of location and previous goal schema), whereas our system only considers actions. We use abstract goals for partial recognition (not reported in this paper), whereas their system only makes full predictions. They also do not do parameter recognition.

Relatively little attention has been given in the literature to goal *parameter* recognition. Most systems either do not do parameter recognition at all (e.g., (Albrecht, Zukerman, & Nicholson 1998; Pynadath & Wellman 2000)), or use logical-based, non-probabilistic reasoning based on a plan library (e.g., (Bauer 1995)). The only other system we are aware of that does probabilistic parameter recognition is (Charniak & Goldman 1993), which includes action parameter values as nodes in the BN as well as the probability that they fill a slot in the plan. Parameter recognition, however, was not the focus of their research, and they simply assume that all objects of a given type are equally likely to fill a goal parameter.

Conclusions and Future Work

We have presented a goal schema recognizer and a goal parameter recognizer that are trainable from a plan corpus. The recognizers are much faster than previous work (linear in the number of goals and observed actions, respectively) and get fairly good results, considering the domain.

For future work, we plan to extend the recognizer to handle hierarchical (decomposition) plans. 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 we still cannot recognize the high-level goal.

As additional future work, we plan to explore the use of AI planners to generate artificial plan corpora to be used for training. The approach we plan to take combines planning and Monte-Carlo simulation to generate plan corpora. The idea is to generate plans stochastically (allowing distributions over different aspects of the planning process, such as the goals, situations and action decompositions). By combining "context-free" Monte-Carlo simulation techniques with richly context-dependent planning algorithms, we hope to obtain a corpus that captures likely user behavior. In addition, this generated corpus has the big advantage that the subgoal hierarchy that generates the observed actions is also known, which should help in recognition of hierarchical plans.

References

- Albrecht, D. W.; Zukerman, I.; and Nicholson, A. E. 1998. Bayesian models for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction* 8:5–47.
- Allen, J.; Byron, D.; Dzikovska, M.; Ferguson, G.; Galescu, L.; and Stent, A. 2000. An architecture for a generic dialogue shell. *Journal of Natural Language Engineering special issue on Best Practices in Spoken Language Dialogue Systems Engineering* 6(3):1–16.
- Bauer, M. 1995. A Dempster-Shafer approach to modeling agent preferences for plan recognition. *User Modeling and User-Adapted Interaction* 5(3–4):317–348.
- Blaylock, N., and Allen, J. 2003. Corpus-based, statistical goal recognition. In Gottlob, G., and Walsh, T., eds., *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, 1303–1308.
- Bui, H. H. 2003. A general model for online probabilistic plan recognition. In Gottlob, G., and Walsh, T., eds., *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*.
- Carberry, S. 1990. *Plan Recognition in Natural Language Dialogue*. ACL-MIT Press Series on Natural Language Processing. MIT Press.
- Charniak, E., and Goldman, R. P. 1993. A Bayesian model of plan recognition. *Artificial Intelligence* 64(1):53–79.
- Horvitz, E., and Paek, T. 1999. A computational architecture for conversation. In *Proceedings of the Seventh International Conference on User Modeling*, 201–210. Banff, Canada: Springer-Verlag.
- Huber, M. J.; Durfee, E. H.; and Wellman, M. P. 1994. The automated mapping of plans for plan recognition. In de Mantaras, R. L., and Poole, D., eds., *UAI94 - Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, 344–351. Seattle, Washington: Morgan Kaufmann.
- Kautz, H. 1991. A formal theory of plan recognition and its implementation. In Allen, J.; Kautz, H.; Pelavin, R.; and Tenenber, J., eds., *Reasoning about Plans*. San Mateo, CA: Morgan Kaufman. 69–125.
- Lesh, N.; Rich, C.; and Sidner, C. L. 1999. Using plan recognition in human-computer collaboration. In *Proceedings of the Seventh International Conference on User Modeling*. Banff, Canada: Springer-Verlag. Also available as MERL Tech Report TR-98-23.
- Lesh, N. 1998. *Scalable and Adaptive Goal Recognition*. Ph.D. Dissertation, University of Washington.
- Pynadath, D. V., and Wellman, M. P. 1995. Accounting for context in plan recognition, with application to traffic monitoring. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, 472–481. Montreal, Canada: Morgan Kaufmann.
- Pynadath, D. V., and Wellman, M. P. 2000. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-2000)*, 507–514.
- Vilain, M. 1990. Getting serious about parsing plans: a grammatical analysis of plan recognition. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 190–197. Boston: AAAI Press.