A Sound and Fast Goal Recognizer

Neal Lesh and Oren Etzioni*
Department of Computer Science and Engineering
University of Washington, Seattle, WA 98195
neal@cs.washington.edu, etzioni@cs.washington.edu
(206) 616-1849 FAX: (206) 543-2969

Abstract

The bulk of previous work on goal and plan recognition may be crudely stereotyped in one of two ways. "Neat" theories — rigorous, justified, but not yet practical. "Scruffy" systems — heuristic, domain specific, but practical. In contrast, we describe a goal recognition module that is provably sound and polynomial-time and that performs well in a real domain. Our goal recognizer observes actions executed by a human, and repeatedly prunes inconsistent actions and goals from a graph representation of the domain. We report on experiments on human subjects in the Unix domain that demonstrate our algorithm to be fast in practice. The average time to process an observed action with an initial set of 249 goal schemas and 22 action schemas was 1.4 cpu seconds on a SPARC-10.

1 Introduction and motivation

Plan recognition (e.g. [Kautz, 1987; Pollack, 1990]) is the task of identifying an actor's plan and goal given a partial view of that actor's behavior. We have focused on identifying the actor's goal. There are several potential applications for an effective goal recognizer. Goal recognition is useful for enhancing intelligent user interfaces [Goodman and Litman, 1992]. Furthermore, a goal recognizer would allow an autonomous agent to provide useful services to the people it interacts with, such as completing their current tasks and offering advice on how to better achieve future goals. This paper describes a goal recognition module. We do not consider how to integrate a goal recognizer into an agent's architecture.

In our scheme, the system observes the actor executing a sequence of actions. The actor does not necessarily know she is being observed (this is keyhole recognition). The system attempts to identify the actor's goal as early as possible. What does it mean to identify the actor's goal? To attribute a goal to an actor is to predict that the actor's actions, both observed and unobserved, will be the execution of one of the plans for that goal. The actor's goal may be a conjunction of various goals such as "Taking out the trash *and* fixing the car".

The following scenario illustrates the sort of conclusions we want our goal recognizer to produce. The observed actor is a computer user, entering commands into a Unix shell. Suppose we observe:

```
>cd /papers
```

There are many plausible goals at this point. The actor might be searching for a particular file. Or perhaps she wants to know how much memory is used by the /papers directory. But her goal is *not* to find a free printer. Nor is she reading her mail. Why bother to change directories for these goals? Changing directories is irrelevant to these goals and we assume the actor does not execute irrelevant commands. If we allowed arbitrarily many irrelevant actions then we could not predict the actor's goal because all the observed actions might be unrelated to her goal. Suppose that we next observe:

```
>ls
paper.tex  paper.ps
```

The second line is the output from ls. The goals we previously rejected are still rejected. Let's reconsider the goal of determining the memory usage of /papers. The optimal approach is to execute du instead of ls. If actors acted optimally, we could reject this goal. But actors do not always act optimally. This actor may next execute ls -l on paper.tex and on paper.ps and add the memory usages of the files together. Thus, ls may be part of a suboptimal plan to determine the memory usage of /papers. Now we observe:

```
>grep motivating paper.tex
```

We now reject the memory usage goal. We also reject the goal of searching for a file named paper.tex because the grep command does not contribute to it. The actor might, however, be looking for a file that contains "motivating" or perhaps for a file that contains "motivating" and is named paper.tex.

In the following sections we articulate the definitions, assumptions, and algorithm we use to express, justify, and produce these conclusions. Although we validate our system in Unix, our algorithm and formal results are domain independent.

"Many thanks to Denise Draper, Terrance Goan, Robert Goldman, Steve Hanks, Henry Kautz, Nick Kushmerick, Diane Litman, Mike Perkowitz, Rich Segal, Tony Weida, and Dan Weld for comments and discussion. Special thanks to Keith Golden and Mike Williamson for their ongoing supply of critique and coffee. This research was funded in part by Office of Naval Research grant 92-J-1946 and by National Science Foundation grant IRI-9357772.

2 Overview of the paper

Our objective is to build a goal recognizer that performs well in large, real domains. We need a way to quickly determine if a goal is *inconsistent* with the observed actions. Informally, a goal is inconsistent with the observed actions if the actor could not possibly have executed the actions as part of a plan to satisfy the goal. To determine consistency, we must reason about *all* plans for each candidate goal. To reason tractably, we borrow techniques for constructing and manipulating graph representations of planning problems, originally developed to produce search control for generative planners [Etzioni, 1991; Smith and Peot, 1993]. By analyzing interactions among the actions, action schemas, and goal schemas in our *consistency* graphs (defined below), we detect cases where no valid plan exists for a goal.

Most plan recognition algorithms run in exponential time and have not been shown to perform well on large problems. In contrast:

- Our algorithm is sound (never rejects a goal *G* unless our assumptions entail that *G* is not the actor's goal) and runs in polynomial time in the size of the input goal recognition problem. This input is smaller than the corresponding input to most plan recognizers.
- Our implementation is fast. We have tested our system on data collected from human subjects in the Unix domain. The average time to process an observed action with an initial set of 249 goal schemas and 22 action schemas was 1.4 cpu seconds.[1]

In our formulation, goal recognition is semi-decidable. It follows that our polynomial-time algorithm is incomplete, i.e. it is not guaranteed to reject every inconsistent goal. In our experiments, however, the algorithm rejects most inconsistent goals.

This paper is organized as follows. Section 3 defines our terms, the input and output of a goal recognizer, and states our assumptions. Section 4 introduces consistency graphs, describes our algorithm, and works through an illustrative example. Section 5 describes our empirical validation. Finally, sections 6 and 7 discuss related work, the limitations of our system, and future work.

3 Problem formulation

We begin with the informal story. The actor constructs and then executes a plan to solve her current goal. Although plans may contain conditionals, the observable behavior that results from the execution of any plan is a sequence of actions. The system observes a prefix of this action sequence. A plan is *consistent* with the observed actions *A* iff that plan has an execution with prefix *A.* A goal is *consistent* with *A* iff there exists a consistent plan for that goal. A key question is what constitutes a plan for a goal? We assume the actor constructs plans without any irrelevant actions. We further assume the actor constructs plan *P* for goal *G* only if *P might* achieve *G* given the actor's beliefs about the world. This assumption suggests the system has access to the actor's beliefs. The system is given, as input, an arbitrary subset of the actor's beliefs. The more of the actor's beliefs the system is given, the more goals it can prove inconsistent.

[1] The data we have collected is publicly available. Send mail to neal@cs.washington.edu for details.

3.1 Planning language

Our formulation is general enough to accommodate many planning languages. We use UWL [Etzioni *et al,* 1992], an extension of the STRIPS language, because it can express information-gathering goals and actions with sensory effects. This is necessary to distinguish between Unix commands such as pwd and cd. In UWL, *states* are sets of literals. A *literal* is a possibly negated atomic formula. The conjunction of literals in a state describes all relevant relationships in that world state. *Models* and *goals* are also sets of literals. The conjunction of the literals in a model or goal is a partial description of a state. A *goal schema* is a set of literals which can contain variables. A schema can be *instantiated* by replacing variables with constants. For example, the goal schema for searching for a file with some name ?n (where ?n is a variable) can be instantiated to form the goal of searching for a file named paper.tex. An *action schema* consists of a name, a precondition set, and an effects set. An *action* is an instance of an action schema, with a unique id number. The CD action schema, for example, can be instantiated into many different actions, such as cd /papers or cd bin. Multiple executions of the same action are distinguished by their id numbers.

Informally, *plans* are programs composed of nested conditionals and actions. For brevity, we will not define plans but instead make use of a function, *Executor,* which acts as an interpreter for plans. *Executor* is a many-to-one mapping that maps a plan and state to the action sequence that results from executing the plan in the state. We often refer to an action being *in* a plan, or one action coming *before* another. Action A_i is in plan *P* iff there exists a state *S* such that *Executor(P,S)* —* *[..,Ai,..]*. If *Ai* and *Aj* are both actions in plan P, then A_1 is *before* A_3 iff in every execution of *P* in which A_j appears, A_1 appears prior to A_j.

3.2 Consistency

A plan is *consistent* with the observed actions if some execution of the plan might produce those observations.

Definition 1 *Plan P is* consistent *with sequence of actions A iff there exists state S such that Executor maps P and S to a sequence of actions of which A is a prefix.*

Consistency of a goal is defined relative to a set of action schema A and a model M. A goal is consistent with the observed actions only if there exists a consistent plan, built out of A, for that goal. A plan for a goal must *potentially-achieve* the goal, given model M, and contain no irrelevant actions. We now define these terms.

Definition 2 *Plan P* potentially-achieves *goal G given model M iff there exists a state S such that M* C *S and G is satisfied by executing P in S.*

If *P* potentially-achieves *G* given *M* then *P* also potentially-achieves *G* given any subset of *M.* This

- A goal recognizer takes a goal recognition problem II as input and returns a set of goal schemas G′.
- The input II = *(A,M,A,G)* where
  - *A* is a sequence of actions. *A* is assumed to be a prefix of the actions executed by the actor.
  - M is a set of beliefs. M is assumed to be a subset of the actor's beliefs.
  - A is a set of action schemas. A is assumed to be a superset of the action schemas the actor plans with.
  - G is a set of goal schemas. The actor's goal is assumed to be an instance of an element in G.
- The output *Q′* is a subset of G such that for every goal schema in G*Q′* there exists an instance of that goal schema G,- and a plan *P* such that:
  - *P* could achieve goal G, given the actor's beliefs (constrained by *M).*
  - *P* contains no irrelevant actions.
  - *P* is composed of actions from A.
  - Some execution of *P* is a sequence of actions with prefix *A.*
- Multiple Goals: The actor's goal may be the conjunction of various goals, such as "Taking out the trash *and* fixing the car *and* writing a paper"

Figure 1: Input/output specification for a goal recognizer.

makes it easy to treat the beliefs *M* in the input II as a subset of the actor's beliefs.

What does it mean for a plan to contain no irrelevant actions? We require that every action in the plan *support* some action in the plan or the goal.[2] We define *support* between actions as follows (the definition of an action supporting a goal is very similar).

Definition 3 *Let A_i and Aj be actions in plan P. A_x* supports *Aj iff A_i is before Aj, A_i has an effect that unifies with a precondition p of Aj, and no action between A_i and Aj in P has an effect which negates p.*

Support is blocked only by an action that *negates* the supporting effect. One upshot is that we allow redundant sensory actions in the actor's plans. In other words, we do *not* assume that the actor remembers everything she learns from executing sensory actions. We can now define consistency for goals.

Definition 4 *Goal G is* consistent *with action sequence A, given model M and action schemas A, iff there exists plan P such that (1) P is consistent with A, (2) every action in P is an instance of a schema in A, (3) P potentially-achieves G given model M, and (4) every action in P supports an action in P or supports G.*

A goal schema is consistent if *any* instance of that goal schema is consistent.

[2]This is a much weaker requirement than that the plan be minimal in order to contain no irrelevant actions.

3.3 I/O of a goal recognizer

A *goal recognition problem II* is a four-tuple *{A, M,* A, G*)* where *A* is an action sequence (the observations), *M* is a model (a subset of the actor's beliefs), A is a set of action schemas (the actor's plan is composed of actions in this set), and G is a set of goal schemas, A *goal recognizer* takes a goal recognition problem and returns G′ C G. Ideally, G′ is the set of goal schemas in G that are consistent with *A,* given *M* and A. Figure 1 summarizes the input/output specification for a goal recognizer.

We view goal recognition as the process of discarding goal schemas from the input set G and returning the remaining, unrejected goals. A *sound* recognizer never discards a consistent goal. A *complete* recognizer discards every inconsistent goal.

When the recognizer returns G", this means "the actor's goal is an instance of a schema in G*.[n] If the recognizer is sound, this conclusion is justified by the following assumption.

Assumption 1 *The actor's goal is consistent, given model M. and action schemas A, with the (full) action sequence the actor executes, of which A is a prefix. Further, the actor's goal is an instance of a schema in* G.

Determining consistency is exponential in the length of the longest plan (that doesn't go through the same state twice). In an unbounded domain, consistency is semi-decidable. In the next section, we describe a sound, polynomial-time, but incomplete goal recognizer.

4 Our goal recognizer

We now present our goal recognizer, i.e an algorithm which takes a goal recognition problem II and returns a set of goal schemas. This goal recognizer is provably sound and runs in polynomial time. In this section, we present our theorems and provide intuitions for why they are true. The full proofs are in [Lesh and Etzioni, 1995]. In section 5, we validate our algorithm using data gathered in the Unix domain.

Our goal recognizer constructs and manipulates a single consistency graph *T* based on the input II. A *consistency graph* is a directed graph in which the nodes are actions, action schemas, and goal schemas. Figure 2 shows a simple consistency graph.

Informally, the consistency graph represents the plans the actor might be executing. The actions in T represent the observed part of the actor's plan. The action schemas in T represent possible unobserved actions. The goal schemas in *T* represent the goals the actor might have. The edges in *T* indicate when one action can support another action or goal. A consistency graph is *correct* if all consistent plans are represented by the graph.

Definition 5 *A consistency graph* (V,£) *is* correct, *relative to the input goal recognition problem* II = *{A, M, A,G), iff the following three properties hold:*
(P I) V *contains every consistent goal schema in* G,
(P2) V *contains action schema* A E A *if an instance of X is in any consistent plan for a goal in* G, *and*
(P3) *S contains the edge (Vi—*Vj) for every Vi,Vj* G V *where V_i (or an instance of V_i) supports Vj (or an instance ofV_j) in some consistent plan for a goal in* G.

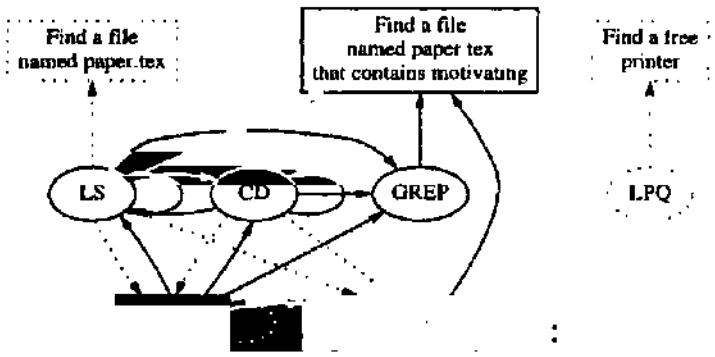

Figure 2: Consistency graph for a small, illustrative example: the goals are in unshaded boxes, action schemas in ovals, and observed actions in shaded boxes. The dotted edges and nodes represent elements that have been removed.

A fully connected graph with a node for every observed action, action schema, and goal schema in Π is trivially correct. The *initialize* function produces such a graph.

Function 1 initialize($\Pi = (\mathcal{A}, M, \Lambda, \mathcal{G})$) ::: *returns a fully connected graph with nodes* $\mathcal{A} \cup \Lambda \cup \mathcal{G}$.

Our goal recognizer maintains a single consistency graph. It uses *initialize* to construct a correct, but large, consistency graph. The algorithm then removes elements from this graph without violating correctness. Removing an action schema from the graph indicates that no unobserved action is an instance of the action schema. For example, removing the LPQ action schema indicates that none of the unobserved (future) actions will be an `lpq` command. Removing a goal schema from the graph indicates the goal schema is inconsistent, i.e. that no instance of that schema can possibly be the actor's goal.

Ideally, we would remove elements until we produced a minimal consistency graph. This graph would be maximally predictive. In general, however, the minimal graph is intractable to compute. Instead, we prune elements from the graph based on linear-time computations. For example, we will show that if there is no path from an observed action to a goal, then that goal must be inconsistent. Thus, we can safely and quickly remove the goals "Find a file named paper.tex" and "Find a free printer" in the graph in Figure 2 because these goals are not connected to the **grep** command. These linear-time computations detect some, but not all, opportunities to remove elements without violating correctness.

Rules remove elements from the graph that are not in any consistent plan. A rule returns $(\mathcal{V}' \subseteq \mathcal{V}, \mathcal{E}' \subseteq \mathcal{E})$ given $(\mathcal{V}, \mathcal{E})$. A rule is *legal* iff it preserves correctness; i.e. it returns a correct graph when given a correct graph.

For example, suppose that no effect of action schema λ_i unifies with any precondition of action schema λ_j. By Definition 3, no instance of λ_i can support any instance of λ_j. Thus, removing edge $\langle \lambda_i \rightarrow \lambda_j \rangle$ from a correct graph Γ does not violate property (P3). Furthermore, removing an edge never violates properties (P1) or (P2). This gives rise to the following theorem:

Theorem 1 *(Matching)*
IF no effect of V_i unifies with any precondition of V_j
THEN $\mathcal{E}' = \mathcal{E}$ - $\langle V_i \rightarrow V_j \rangle$ *is legal.*

Several edges are missing from the consistency graph shown in Figure 2, such as (LPQ→GREP) and (GREP→LPQ). These edges were removed by the Matching Rule.[3]

We will now present several, but not all, of our rules. The Order Rule removes an edge from A_i to A_j if A_i does not precede A_j. The legality of this rule follows from the fact that action A_i can support action A_j only if A_i comes before A_j.

Theorem 2 *(Order)*
IF $(V_i, V_j \in \mathcal{A}) \wedge \neg(\mathcal{A} = [.., V_i, .., V_j, ..])$
THEN $\mathcal{E}' = \mathcal{E}$ - $\langle V_i \rightarrow V_j \rangle$ *is legal.*

If we observed `cd /papers` and then `ls`, the Order Rule would remove the edge from `ls` to `cd /papers` since `ls` cannot support any previous action. The Order Rule only removes edges between actions, not action schemas. The edge (`ls`→CD), for example, is not removed because `ls` may support some future, unobserved instance of the CD action schema.

The Prefix Rule removes edges from action schemas to actions. Recall that action schemas in Λ represent the unobserved actions, and actions in $\mathcal{A}$ represent the observed actions. No unobserved action can support an observed one, because the observed actions precede all the unobserved actions.

Theorem 3 *(Prefix)*
IF $(V_i \in \mathcal{A}) \bigwedge (V_j \in \Lambda)$ THEN $\mathcal{E}' = \mathcal{E}$ - $\langle V_j \rightarrow V_i \rangle$ *is legal.*

Without the Prefix Rule, the observations are treated as an ordered subset, rather than a prefix, of the actor's behavior.

To motivate the next rule, suppose that action A_1 has effects p and q and action A_2 has effect $\neg p$. By Definition 3, effect p in A_1 cannot support any action after A_2. We can thus remove edges from A_1 that rely on effect p.

Theorem 4 *(Clobber)*
IF $\mathcal{A} = [..V_i, .., V_j, .., V_k..]$ *and* V_j *has an effect which negates the effect in* V_i *that supports the precondition of* V_k THEN $\mathcal{E}' = \mathcal{E}$ - $\langle V_i \rightarrow V_k \rangle$ *is legal.*

Due to space limitations, we present only two goal removal and one action schema removal rules. The Goal Connection Rule discards a goal if there is not a path in Γ from every observed action to that goal. Recall that edges indicate support. If some action A is not connected to a goal G, then A cannot support, directly or indirectly, goal G. By definition, A cannot be in any plan for G, and thus G is inconsistent.

Theorem 5 *(Goal Connection)*
IF $(G \in \mathcal{G}) \wedge (\exists_{A_i \in \mathcal{A}}$ *s.t.* $\neg$Connected$(A_i, G))$
THEN $\mathcal{V}' = \mathcal{V}$ - G *is legal.*

The Goal Connection Rule removes "Find a file named paper.tex" and "Find a free printer", in Figure 2, because **grep** is not connected to them. Also, we automatically remove edges that point to removed nodes.

The Impossible Conjunct Rule leverages our assumption that the actor's plan possibly-achieves the actor's goal given the actor's beliefs about the world. On the

[3] The Matching Rule is legal even if V_j is a goal, because we define the *preconditions* of a goal G (which is a set of literals) to be the members of G.

strength of this assumption, we can reject a goal if no plan exists which could achieve it, given the actor's beliefs. A simple case where this arises is when a conjunct of some goal is false in the model M and not supportable by any effect of any schema in $\Lambda \cap \mathcal{V}$

Theorem 6 (Impossible Conjunct)
I F $g \in G$ *is* isupportable and false in M
THEN $\mathcal{V}' = \mathcal{V}$ - G is legal.

The Obsolete Rule leverages our assumption that every action supports, directly or indirectly, the goal. If some action schema λ is not path-connected to any goal in Γ then no instance of λ can support any goal in Γ. Since Γ is correct, it contains every consistent goal. Thus λ cannot be in any consistent plan for a consistent goal.

Theorem 7 *(Obsolete)*
IF $(V_i \in \Lambda) \wedge (\forall_{g \in (\mathcal{V} \cap \mathcal{G})} \cdot \neg\text{Connected}(V_i, g))$
THEN $\mathcal{V}' = \mathcal{V}$ - V_i *is legal.*

The following algorithm is a sound goal recognizer that runs in polynomial time in the size of II.

Function 2 recognize(II) ::: Apply the rules to quiescence on initialize(II). Return all goal schemas in the resulting graph.

Theorem 8 Recognize is sound and polynomial-time in the size of XI.

The initialize function produces a correct graph, and since we only apply legal rules, the final graph is correct as well. A correct graph contains every consistent goal schema. Thus our algorithm returns every consistent goal schema and therefore is sound.

The initial graph contains \G U Au A\'^{2} elements. In every iteration, we potentially apply every rule to every element. Each rule can be applied to an element in linear time in the size of the graph. Our algorithm halts as soon as applying all the rules fails to remove anything. Thus, the maximum number of iterations is the number of elements in the initial graph. An upper bound for the worst case running time is thus k x \Q U A U A|^6 where k is the number of rules. This is a loose upper bound, intended only to show that our algorithm is polynomial.

Our actual implementation is optimized for our current rule set. We have analyzed the dependency relationships among our rules, and fire only a subset of the rules in each iteration. Additionally, we often apply many rules in a single procedure. We test, for example, the connectedness between an observed action and all goals with one procedure rather than a call for each goal. This is roughly $|\mathcal{G}|$ times faster than checking every goal separately. Furthermore, although the above algorithm processes all observed actions at once, our actual system is incremental. We fold new actions into the processed graph rather than re-initialize the graph every time a new action is observed. When we observe a new action A_n, we add A_n to $\mathcal{V}$, fully connect A_n to and from every node and then apply our rules to this new graph.

4.1 Example trace

Now we present a sample trace. The input to the recognizer is II = (A, M, A, 0) where A = [cd /papers, grep motivating paper.tex], Λ contains representations of CD, LS, GREP, and LPQ, $\mathcal{M}$ is empty (we know none of the actor's beliefs), G contains G\ = "Find a file named paper.tex", G_2 = "Find a file named paper.tex that contains motivating", and G_3 = "Find a free printer."

The initialize function produces a graph similar to the one in Figure 2 except that it is fully connected.

In the first iteration, the Matching Rule removes 61 edges, resulting in the graph shown in Figure 2. The Order Rule removes edge ⟨**cd /papers→cd /**papers). The Prefix Rule removes e_d g⟨CD→**cd** / e r s), (LS→**cd** /papers), ⟨CD→**grep**⟩, and ⟨LS→**gr**ep).

In the second iteration, Goal Connection removes G_1, G_3, and edges ⟨LS→G_1⟩ and ⟨LPQ→G_3⟩) because they point to removed goals.

In the third iteration, the Obsolete Rule removes LPQ.

In the fourth iteration, no rules fire. Set $\{G_2\}$ is returned.

5 Experimental validation

In this section, we describe how we collected our data, how we converted it into goal recognition problems, and how our system performed on these problems.

We gather the raw data as follows. After reading some instructions, the subjects (students in our department) are given English goal descriptions. The subjects try to solve each goal by executing Unix commands. For each task, the subjects indicate success or failure. We prohibit subjects from using certain constructs and commands such as pipes and the command awk.

We converted this raw data into goal recognition problems. To generate the observed actions A, we matched each observed Unix command to an instance of our action schema for that command. We filtered out commands we considered to be typos. We let A be our action schemas for 22 Unix commands, including all those used by our subjects. We let $\mathcal{M} = \emptyset$, indicating that we know nothing about the actor's initial beliefs. This is the most conservative setting. The goal space $\mathcal{G}$ consisted of 249 goal schemas: 129 file-search goal schemas (goals of locating a file that has some qualities), 15 assorted (non file-search) goals, and the 105 pairs of these 15 goals. We evaluated our goal recognizer on the following goals:

- (G_1) Find a file named "core".
- (G_2) Find a file that contains "motivating" and whose name ends in ".tex".
- (G_3) Find a machine that has low (< 1.0) load; and determine if Oren Etzioni is logged into the machine named chum.
- (G_4) Compress all large ($>$ 10,000 bytes) files in the Testgrounds subdirectory tree.

Goals G\ and G_2 are two of the 129 file-search goals. Goal G3 is one of the 105 pairs of the assorted goals. This goal demonstrates our ability to handle interleaved plans for multiple goals. Goal G_4 is one of the 15 assorted goals. Table 1 summarizes our results. An update is when one observed action is processed. The length of the plan is the number of actions the subjects executed to achieve the given goal. The remaining goals are the goals still in the graph after the last update.

goal	cpu seconds per update	length of plan	goals remaining	updates until subset convergence
G_1	1.616	2.25	155	--
G_2	1.643	16	37	4
G_3	.648	3.0	1	1
G_4	1.610	20.5	15	5.5

Table 1: Summary of results. An update is processing one observed action. The length of the plan is the number of actions executed to achieve the given goal. We ran our algorithm, coded in Lisp, on a SPARC-10. Each number is the average over data gathered from two to four subjects.

Our goal recognizer performed very well on these data. The average time to process an observed action was 1.4 cpu seconds on a SPARC 10 by code written in lisp. Although the goal recognizer is incomplete in general, it detected every inconsistency between the goals and observations in our experiments. Thus, our algorithm solved the goal recognition problem, *as we have formulated it,* very thoroughly and very quickly.

But did it work? Did our mechanism recognize people's goals? In some sense, goal recognition occurs when the recognizer returns a single, consistent goal. This rarely occurred in our experiments. For example, for goal $_{G1}$, over half the goals remain. The subjects only executed cd and ls until they found core. These commands both support almost all file related goals. Thus, almost all file related goals are consistent with the entire plans executed to solve goal *G*\.

A common assumption in many plan recognition paradigms is that the actor's actions will eventually serve to distinguish a single plan (or goal). Our investigations suggest this is unlikely to be true in the Unix domain, even with relatively small sets of possible goals. We view our goal recognizer, which quickly prunes out the inconsistent goals, as a useful module. The next step might be to assign probabilities to the remaining goals. In [Lesh and Etzioni, 1995] we propose a very different solution based on version spaces [Mitchell, 1982]. We view goals as hypotheses. When a single, strongest consistent goal-hypothesis exists, we know that achieving this goal will benefit the actor. Due to space limitations, we describe a special case of this approach, *subset convergence,* that works well on these problems.

We define *subset convergence* to occur when all the goals in the space share a common (non-empty) subset. Subset convergence is useful because an agent might be able to make use of the fact that some goal *G'* is *part of* the actor's goal. The last column in Table 1 indicates when subset convergence occurred for each goal (except goal G, for which subset convergence did not occur).

Consider goal G_2. All but 37 goals are rejected when a grep is observed. The remaining 37 goals all involve searching for a file that contains some word and, possibly, has some other characteristics (e.g. name ends in ".tex"). The goal of looking for a file that contains some word is a subset of all 37 of these goals. For G_2, subset convergence occurred, on average, after four actions were observed, which was an average of 12 actions before the subjects completed the task.

For goal G_3, the subject's first command always indicated that *part of* their goal either was to find a machine with a low load, or to determine if Oren was logged into chum. Thus, subset convergence occurred immediately. By the second or third command, the goal space converged to goal G_3.

For goal G_4, the 15 unrejected goals are *G4* itself and the 14 pairs of the assorted goals that include *G4.* Nothing the subjects did to achieve G_4 was contrary to (or inconsistent with) the goal of compressing all large files *and,* for example, finding a free printer. Again, subset convergence detects, early on, that all 15 goals include the goal of compressing the large files.

6 Related work

Most plan recognizers (e.g. Kautz's) require, as input, a plan or event hierarchy which consists of top-level goals, primitive actions, and composite or complex actions. Our input, however, differs significantly. Essentially, we take only the goals (input G) and primitive actions (input A). Our definition of what constitutes a valid plan for a goal replaces the complex actions. Under our formulation, the goal recognizer must consider how the low level actions can be composed into plans. Eliminating the complex actions is significant in that there may be up to $2^{|A|}$ complex actions in the hierarchy. Although our input is more compact, it is less expressive; we do not allow arbitrary constraints to be placed between steps in plans. We do, however, allow arbitrarily long plans which an acyclic plan hierarchy does not.

A rarely duplicated feature of Kautz's theory and system is the ability to recognize concurrent, interleaved plans. Kautz assumes that actors execute the minimum number of consistent plans. We can proceed similarly if we assume that the concurrent execution of plans $P_1...P_n$ for goals $G\...G_n$ is always the execution of some single plan *P* for $G\f\ ...\ A\ G_n$- The technique to recognize interleaved goal-solving is to first run our algorithm, as normal, on the given goal space G. If all goals are rejected then run recognition on all pairs of goals, G^2. If this space collapses, run recognition on G^3. And so on. This approach is not polynomial or sound because we approximate consistency, *if we allow arbitrary numbers of plans (or goals) to be interleaved.* We believe, however, that people rarely interleave large numbers of plans. If we assume that actors only pursue at most, say, 3 or 4 goals simultaneously then our technique, which can terminate at G^6 or G^4, becomes polynomial and sound.

There has been some work (e.g. [Pollack, 1990]) on recognizing invalid plans. We allow some invalid plans, because we allow all plans which could achieve the goal based on the actor's incomplete model of the world. Some of these plans will not achieve the goal when executed from the actual world state. We cannot recognize plans built out of incorrect models of the action schemas, as Pollack's system can. Her system does not consider all allowable plans, as ours does, but instead searches for a good explanatory plan.

There is some work on trying to select the best or most probable plan or combination of plans (e.g. [Charniak and Goldman, 1991]). Our work complements this re-

search; our recognizer can produce the consistent goals which can then be subjected to more expensive probabilistic analysis. [Weida and Litman, 1992] extend term subsumption to include plan recognition, motivated by the need to organize large numbers of plans, much like our desire to handle large domains. [Vilain, 1990] describes a polynomial-time plan recognizer based on grammatical parsing. His system is sound and complete on restricted classes of plan hierarchies unlike ours which approximates consistency relationships based on an exponentially large plan space. [Bauer *et al,* 1993]'s definition of when a plan can be *refined* to include actions resembles our definition of a plan being consistent with actions, but their computational approach is quite different from ours.

7 Critique and future work

Our algorithm requires polynomial time in size of the input. A more sophisticated algorithm, described in [Lesh and Etzioni, 1995], runs in time linear in the number of input goals (though still polynomial in |A|). But reasonable goal spaces may be exponentially large in relevant features of the domain, such as the number of predicates. Our solution is based on version spaces [Mitchell, 1982]. We view goals as hypotheses and explicitly compute only the strongest consistent hypotheses and the weakest consistent hypotheses. These two boundaries compactly represent the set of all consistent goals. In [Lesh and Etzioni, 1995], we identify a class of goals such that we can determine the consistency of 2^n goals by explicitly computing consistency on only n goals.

Currently, we strongly leverage our assumption that every action in the actor's plan supports another action or the goal. On this basis we reject the goal of finding a free printer if we observe cd /papers. However, if we completely model our domain, then most actions can contribute to most goals. If the actor is searching for a file that contains printer names then cd /papers can (indirectly) support finding a free printer. The problem is not that we fail to recognize this obscure plan but that we fail *because* we don't model the world well. Eventually, we will need a stronger constraint than our current one that every action support another action or the goal.

On the other hand, our approach is sensitive to noisy or spurious actions. We assume every observed action is part of a goal-directed plan. This may not adequately capture the role of certain actions such as returning to one's home directory or mopping one's brow. We are currently exploring the possibility of learning which actions are regularly spurious by observing the actor over a long period of time. These actions could then be filtered out from the observations.

We have not yet addressed several additional issues. Recall that the input observations are actions in our formal action language. But how do we automatically produce, for example, an instance of the CD action schema from the observable string cd /papers? What if the actor executes a command which fails? Furthermore, how do we know when the actor finishes one task and begins another? We believe that our goal consistency framework and experimental apparatus puts us in good position to address these issues.

Although our algorithm and formal results are domain-independent, this does not guarantee that our goal recognizer will be effective in every domain. Our case study in Unix indicates that our goal recognizer performs well there. We believe these results suggest that our approach will also work well in various software domains. More generally, our approach is particularly well suited to two classes of goals. First, *conjunctive search goals,* such as the goal of looking for a file that is large, that has not been touched for a month, etc. Second, *conjunctive set goats,* such as compressing all files that are large, that have not been touched for a month, etc. Plans for both classes of goals can be very long, and thus task completion will be especially useful.

References

[Bauer e*t al,* 1993] M. Bauer, S. Biundo, D. Dengler, J. Kohler, and Paul G. Phi-a logic-based tool for intelligent help systems. In *Proceedings of the 13th International Joint Conference on Arttfictal Intelligence.* 1993.

[Charniak and Goldman, 1991] E, Charniak and R. Goldman. A probabilistic model of plan recognition. In *Proc. 9th Nat. Conf. on A.I.,* volume 1, pages 160-5, July 1991.

[Etzioni *et al.,* 1992] O. Etzioni, S. Hanks, D. Weld, D. Draper, N. Lesh, and M. Williamson. An Approach to Planning with Incomplete Information. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning,* October 1992.

[Etzioni, 199l] Oren Etzioni. STATIC: A problem-space compiler for prodigy. In *Proc. 9th Nat. Conf. on A.I.,* 1991.

[Goodman and Litman, 1992] B. Goodman and D. Litman. On the interaction between plan recognition and intelligent interfaces. In *User Modeling and User Adapted Interaction,* volume 2, no. 1-2, pages 83-115, 1992-

[Kautz, 1987] H. Kautz. *A Formal Theory Of Plan Recognition.* PhD thesis, University of Rochester, 1987.

[Lesh and Etzioni, 1995] N. Lesh and O. Etzioni. A sound, fast, and empirically-tested goal recognizer based on version spaces. Technical report, Draft. University of Washington, 1995.

[Mitchell, 1982] T. Mitchell. Generalization as search. *Artificial Intelligence,* 18:203-226, March 1982.

[Pollack, 1990] M. Pollack. *Plans as Complex Mental Attitudes,* pages 77-101. MIT Press, Cambridge, MA, 1990.

[Smith and Peot, 1993] D. Smith and M. Peot. Postponing threats in partial-order planning. In *Proc. 11th Nat. Conf. on A.I.,* pages 500-506, June 1993.

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

[Weida and Litman, 1992] R. Weida and D. Litman. Terminological Reasoning with Constraint Networks and an Application to Plan Recognition. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning,* October 1992.