Scaling up goal recognition

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

Traditionally, plan and goal recognition has investigated examples in which the number of plans and goals is less than one hundred. In this paper, we address three challenges to scaling up goal recognition. First, hand coding the plan library is a tedious if not impractical method of generating massive libraries. Second, previous representations for plan libraries are exponentially large in the number of goal predicates. Third, existing algorithms consider each candidate goal which is prohibitively slow. Our solution to the first problem is to automatically construct plan libraries with *biases* which are explicit assumptions about the type of plans and goals people have. Our solution to the second problem is to represent the consistent goals with a compact, *version space* representation. Our solution to the third problem is a goal recognition algorithm that can quickly update this compact representation given an observed action. Experimentally, in both software and kitchen domains, we show our prototype implementation to be fast on libraries containing thousands of goals. Analytically, we identify a class of goals for which our algorithm is logarithmic in the number of goals.

1 Motivation

Plan recognition is often studied in domains in which people's behavior is characterized by fewer than 100 plans and goals, such as the traffic [Pynadath and Wellman, 1995] and student-scheduling [Carberry, 1990] domains. Our objective is to apply goal recognition, a special case of plan recognition, to domains in which there are tens of thousands of possible goals and hundreds of plans per goal.

The key factor in scaling up plan or goal recognition is the size of the plan library, which contains the plans and goals the actor might have. Massive plan libraries occur because the library can be exponential in the number of primitive actions and goal predicates in the domain. Just as many sentences can be formed from a few words, many plans can be formed out of a few actions and many goals can be formed out of a few predicates. We address three challenges to recognition:

1. How can we generate the plan library?
2. How can we represent plan libraries compactly?
3. How can we perform fast recognition?

In previous work, the libraries were hand coded. For each new domain, a human expert must describe all plans and goals the observed actor might have. Since hand coding libraries with tens of thousands of goals is an enormous knowledge acquisition task, it is desirable to automate construction of the plan library.

Constructing the library, however, is a delicate task. Automating construction requires articulating the principles that govern which plans and goals belong in the plan library. [Weida and Litman, 1992] and [van Beek, 1996], among others, state that the library must be complete, i.e. it must include every plan and goal the actor might execute. We show that the quality of recognition depends on the content of the plan library in another, more subtle way. In particular, the presence of extraneous plans or goals in a complete plan library can increase the number of observations required to identify the actor's plans and goals.

The second issue we address, compact representation, is also particularly desirable for massive libraries because it is not feasible to explicitly store each plan and

*Many thanks to Paul Barton-Davis, Mathias Bauer, Sandra Carberry, Keith Golden, Steve Hanks, Diane Litman, Mike Perkowitz, Tony Weida, Dan Weld, and Mike Williamson for comments and discussion. This research was funded in part by Office of Naval Research grant 92-J-1946, by ARPA / Rome Labs grant F30602-95-1-0024, by a gift from Rockwell International Palo Alto Research, and by National Science Foundation grant IRI-9357772.

goal. The library has been stored in a variety of compact representations including context-free grammars [Vilain, 1990], belief networks [Huber *et al.*, 1994], message passing networks [Lin and Goebel, 1990], and plan-operator graphs [Lesh and Etzioni, 1995]. These representations store plans implicitly but store goals explicitly and contain pairwise links between goals. Thus, the size of these representations grows quadratically in the number of candidate goals which in turn grows exponentially in the number of goal predicates.

Our third challenge, fast recognition, is important because recognizers must respond quickly to a new observation in order to support real-time interaction with people. [Vilain, 1990] and [Lin and Goebel, 1990] imposed restrictions on the plan language to achieve recognition that is linear in the size of an input plan hierarchy $\mathcal{H}$. Since $|\mathcal{H}|$ is quadratic in the number of goals, the recognition algorithms are not sufficiently fast for large domains. Furthermore, plan steps must be totally ordered and cannot be shared or interleaved. This increases $|\mathcal{H}|$ and thus slows recognition considerably. For example, one partially ordered plan of polling five printers to find a free printer corresponds to 120 totally ordered plans. Additionally, since plans for individual goals cannot be interleaved, each candidate *combination* of goals that people might pursue simultaneously must be explicitly stored in $\mathcal{H}$.

We import ideas and techniques from machine learning to meet these three challenges.

2 Overview

Our approach to the above challenges follows from the observation that concept learning (e.g. [Mitchell, 1982, Shavlik and Dietterich, 1990]) is structurally similar to plan recognition. Concept learners select a concept that describes, or fits, a large set of data on the basis of seeing a small portion of that data. Plan recognizers select a plan or goal that describes, or fits, the actor's behavior based on seeing a small portion of that behavior. For example, a concept learner might generalize from "Nixon lied" and "Reagan lied" to "male Republican presidents lie" or "politicians lie". Similarly, a plan recognizer might generalize from observing a user execute the Unix commands `finger smith@alpha` and `finger smith@beta` to the conclusion that the computer user is searching for the machine that Smith is currently logged onto.

The challenges posed above been solved in concept learning. Concept learners do not require an explicitly enumerated concept library but instead receive a set of primitives. Learners construct their own hypothesis space using syntactic *biases*, or rules for composing predicates into hypothesis. We automate construction of the plan library by using plan and goal biases, or assumptions about what *type* of plans and goals people have. We can define, for example, a plan library that contains all optimal plans for goals of length two, or one that contains all plans that achieve goals formed by conjunction over the input predicates.

Concept learners do not explicitly store each candidate hypothesis. We borrowed the version space representation [Mitchell, 1982] of hypothesis spaces from concept learning. We establish a partial-order on goals, using the generality relationship, and store only the boundaries between the consistent and inconsistent regions of the goal space. We identify a special case of conjunctive search goals for which our representation remains a constant size. In a more general case, our experiments show that it remains several orders of magnitude smaller than the set of consistent goals. In one case, boundaries that contains on average only 100 goal combinations represents a set that contains on average 500,000 consistent goal combinations.

We adapted Mitchell's version space algorithm to produce the BOCE goal recognizer. We provide the bias-specific functions the algorithm needs for initializing the boundaries and quickly updating the boundaries given a new observed action. BOCE can infer from the boundaries that some goal is, is part of, or is sufficient for one of the actor's goals. BOCE calls RIGS [Lesh and Etzioni, 1995], to determine the consistency of individual goals. RIGS represents all consistent plans for each consistent goal using consistency graphs, which are based on plan-operator graphs [Etzioni, 1991, Smith and Peot, 1993, Etzioni, 1993]. In the special case mentioned above, BOCE processes observed actions in time that is logarithmic in the number of goals. Experimentally, in two domains, we show BOCE can process a plan library that contains over a million goals in less than 16 cpu-seconds per observed action.

We sacrifice expressiveness to automatically construct the library and enable fast recognition. Automatic construction is not appropriate for all domains. Hand coded libraries can be finely tuned to the actors' behavior. Our approach will work well if the actors' behavior is sufficiently structured so that fine tuning is unnecessary. Additionally, in very large domains, automatic construction may be the only viable option.

The rest of this paper is organized as follows. We define our terms and the I/O of recognition in section 3, automatic construction in section 4, our representation in section 5, and our algorithm in section 6. We discuss an example in section 7, the scope of our approach in section 8, our experiments in section 9, and related work and conclusions in sections 10 and 11.

3 Definitions

We now introduce our terms and specify the inputs and outputs of recognition. Our formulation can accommodate many planning languages (e.g. STRIPS [Fikes and Nilsson, 1971]). The language must define

the concepts of a predicate being *in* a goal, an action being *in* a plan, one action coming *before* another in a plan, an action being an *instance* of an action schema, one plan being a *subplan* of another, and for one goal to *entail* another. Informally, goal g_i entails goal g_j if every state in which g_i is true, g_j is also true. An example goal is (ISA printer x) $\wedge$ (FREE x), where x is a free variable, printer is a constant, and ISA and FREE are the predicates.

Informally, the actor has some goals and constructs a plan for each goal. The plans might share steps or be interleaved. The actor then executes these plans, producing a series of observable actions. Let $\mathcal{GS} = \{g_1, ..., g_n\}$ be the set of goals the actor currently has and $\mathcal{PS} = \{p_1, ..., p_n\}$ be the set of plans the actor is currently executing: the actor is executing plan p_i for goal g_i. A goal is *necessary* (for one of the actor's goals) if it is entailed by a goal in $\mathcal{GS}$, and *sufficient* if it entails a goal in $\mathcal{GS}$. A plan p is *necessary* if p is a subplan of a plan in $\mathcal{PS}$, and *sufficient* if a plan in $\mathcal{PS}$ is a subplan of p.

An *actor model* is a tuple $\langle \Lambda, \Omega, K \rangle$ where Λ is a set of action schemas, Ω is a set of predicates, and K is an integer. Informally, the actor is assumed to construct as many as K plans and goals out of the primitives in Λ and Ω, respectively. As shown in figure 1, a *plan recognizer* is a function which takes an action sequence $\mathcal{A}$ and an actor model $\langle \Lambda, \Omega, K \rangle$ as input and outputs Nec(x) or Suff(x), where x is a plan or goal, indicating x is necessary or sufficient. A *goal recognizer* outputs only necessary goals and sufficient goals.

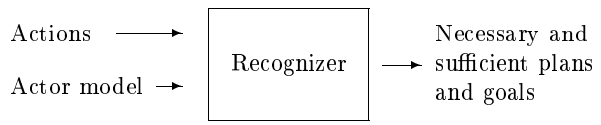

Figure 1: I/O for biased plan recognition

In section 4.2 we show how the recognizer constructs a plan library from the primitives Λ and Ω. The plan library $\mathcal{PL}$ is a set of plan goal pairs $\langle p_i, g_i \rangle$. Each pair $\langle p_i, g_i \rangle \in \mathcal{PL}$ indicates the actor might execute plan p_i for goal g_i. The plan library is *complete* if the actor never executes plan p_i for goal g_i unless $\langle p_i, g_i \rangle \in \mathcal{PL}$.

The recognizer reasons about which combinations of up to K goals in $\mathcal{PL}$ could account for the observed actions $\mathcal{A}$. Action sequence $\mathcal{A} = a_1, ..a_n$ is a *partial interleaved execution* of plans $p_1, .., p_m$ only if (1) for all $1 \leq i \leq n$ there exists some $1 \leq j \leq m$ such that a_i is in p_j and (2) for all $1 \leq i < k \leq n$ and $1 \leq j \leq m$, if a_i and a_k are both in plan p_j then a_i is before a_k in plan p_j. A set of goals $g_1, ..., g_m$ is *consistent* with action sequence $\mathcal{A}$ if there exists plans $p_1, ..., p_m$ such that for all $1 \leq i \leq m$, $\langle p_i, g_i \rangle \in \mathcal{PL}$ and $\mathcal{A}$ is a partial, interleaved execution of plans $p_1, ..., p_m$.

We make the following assumptions.

A1. Action sequence $\mathcal{A}$ is a partial, interleaved execution of the plans $\mathcal{PS}$ the actor is executing.

A2. $\mathcal{PL}$ is complete.

A3. $1 \leq |\mathcal{PS}| = |\mathcal{GS}| \leq K$.

A4. Every action in every plan in $\mathcal{PS}$ is an instance of an action schema in Λ, and every predicate in every goal in $\mathcal{GS}$ is in Ω.

Assumption A1 is important because we cannot recognize an actor's plans from her actions if those actions are not part of her plans. Assumption A2 is important because identifying an actor's plans or goals is essentially the process of eliminating *all* alternative hypotheses, which requires the complete set of possible hypotheses. Assumption A3 is important because, without it, every action might be part of a different plan. Only by observing multiple actions from the same plan can we narrow down the actor's possible plans or goals. Assumption A4 allows the recognizer to construct $\mathcal{PL}$ out of primitives in sets Λ and Ω.

4 Automated construction

We now discuss the content of the plan library and show how to construct a library with biases.

4.1 Extraneous plans and goals

It is often stated (e.g. [Kautz, 1987, Weida and Litman, 1992]) that the plan library must contain every plan and goal the actor could have. It might, then, be tempting to include every conceivable plan and goal in the library. In this section, we show that in addition to being complete, the library should also exclude extraneous plans and goals. A plan or goal is *extraneous* if the actor never executes the plan or has the goal.

First, we review why the plan library should be complete. If the library is incomplete, then the recognizer will produce false conclusions. For example, consider a simple plan library - a cookbook. Suppose we know Smith is cooking one recipe from *The Best Of Lord Krishna's Cuisine* cookbook. After observing Smith slice okra and prepare ghee we conclude that Smith must be cooking Okra Supreme, the only recipe in the *The Best of Lord Krishna's Cuisine* that calls for ghee and sliced okra. Unbeknownst to us, however, Smith is actually using the complete *Lord Krishna's Cuisine* cookbook which includes 8 recipes that call for sliced okra and ghee. Smith's actual meal, Okra Delight, is not in *The Best Of Lord Krishna's Cuisine*.

Now consider the reverse case: we think Smith is cooking from *Lord Krishna's Cuisine* but in fact he is using *The Best Of Lord Krishna's Cuisine*. Our plan library is complete but includes extraneous plans. Again suppose we observe Smith slice okra and prepare ghee.

With the right cookbook, we would know he must be cooking Okra Supreme. But, as it is, there are 8 possible explanations in the cookbook we *think* Smith is using. Thus, the existence of extraneous recipe's prevents us from identifying Smith's goal.

4.2 Bias

Bias fills the gap between the information given to the recognizer (primitives) and the information needed to perform recognition (a complete plan library). *Biases* are grammars for composing primitive actions and predicates into plans and goals. Formally, a *goal bias* GB is a function from a set of predicates to a set of goals; $GB(\Omega)$ refers to the set of goals the actor might compose from the set of predicates Ω. A plan bias PB indicates what plans the actor might construct out of a given set of actions for a given goal. Thus, $PB(g, \Lambda)$ refers to the plans the actor might construct for goal g out of instances of the action schemas in Λ. The plan recognizer's plan library $\mathcal{PL}$ $=\{\langle p, g\rangle \mid g \in GB(\Omega) \wedge p \in PB(g, \Lambda)\}$.

In section 4.1, we showed that recognition suffers if $\mathcal{PL}$ excludes plans and goals the actor does have or includes plans and goals the actor never has. For this reason, the biases should not over-constrain or under-constrain $\mathcal{PL}$. For example, consider the plan bias that admits only optimal plans into $\mathcal{PL}$. If people execute suboptimal plans then this bias over-constrains the library. As described above, this incomplete library will lead to erroneous conclusions. The recognizer might conclude that the observed actor is eating alone when it observes her carry one plate into the dining room, when in fact the actor's actual yet suboptimal plan is to make multiple trips to the kitchen. Conversely, consider the goal bias that allows all conjunctions and disjunctions of goals in Ω. This would under-constrain $\mathcal{PL}$. The observed actions $a_1, .., a_n$ would be consistent with all goals of the form $g_1 \vee ... \vee g_n$ where g_i is an effect of a_i or even an affect of some action a_j where a_i is part of a plan for executing a_j. There would be too many consistent explanations for any given sequence of actions, and the recognizer would never attribute goals to the actor. We currently use the following biases:

Definition 1 (Directed weak) *The* directed weak plan bias *maps* Λ *and* g *to the set of all plans* p_i *s.t. (1) all actions in* p_i *are instances of action schemas in* Λ*, (2) every action in* p_i *supports another action in* p_i *or supports* g *and (3)* p_i *achieves* g *if executed in some world state* s.[1]

Definition 2 (Conjunctive search) *The* conjunctive search goal bias *maps* Ω *to the set of all goals, that consist of predicates in* Ω*, of finding an object that has a conjunction of properties.*

The directed-weak plan bias is flexible in that it allows multiple actions to support the same precondition and allows plans that do not necessarily achieve their goal. Directed-weak plans can contain redundant actions, such as executing a `pwd` command after a `cd` command. Since it allows redundant sensory actions, the directed-weak plan bias does not assume that people remember everything they learn from executing information-gathering commands. This flexibility is important for recognizing the behavior of people. For example, in the Unix domain, we have observed that people often execute long sequences of actions when shorter actions would have sufficed (such as a sequence of `ls`'s and `cd`'s rather than a `find`), and people occasionally give up on tasks that are solvable.

The conjunctive-search goal bias covers an important set of goals because they tend to require long plans. Helping people with long plans is more useful than helping them with something they could do quickly by themselves. Of course, this goal bias only covers a subset of the goals people might actually have. As described in section 6.2, goal biases can be easily combined together. Thus, we can address various classes of goals separately and combine the solutions together to cover a wide range of behavior. We further discuss these biases in section 7.

5 Compact representation

We now describe our implicit representation for plan libraries that is well suited for fast goal recognition.

We represent only the goals or combinations of goals that are consistent (defined in section 3) with the actions observed so far. We borrow Mitchell's term *version space* to refer to the consistent combination of goals. The version space shrinks as actions are observed and more goals become inconsistent. For example, the goal of finding a free printer would be rejected if we observed `compress foo.tex`.

The initial version space $\mathcal{VS}(\emptyset)$ contains all combinations of up to K goals in $\mathcal{PL}$. Recall that the integer K is an input in the actor model which indicates the maximum number of goal the actor might have. Since a conjunction of goals is itself a goal, for convenience we equate having goals $\{g_1, ..., g_n\}$ with having a single conjunctive goal $g_1 \wedge g_2 ... \wedge g_n$. Formally, $\mathcal{VS}(\emptyset) = \{g_1 \wedge ... \wedge g_n \mid (n \leq K) \wedge (\forall_{1 \leq i \leq n} \exists p, \langle p, g_i\rangle \in \mathcal{PL})\}$. $\mathcal{VS}(\mathcal{A})$ contains the goals consistent with $\mathcal{A}$. Formally, $\mathcal{VS}(\mathcal{A} \neq \emptyset) = \{g \in \mathcal{VS}(\emptyset) \mid g \text{ is consistent with } \mathcal{A}\}$. An important lemma that follows from our assumptions is that $\mathcal{VS}(\mathcal{A})$ will contain the actor's current goals.

Lemma 1 *The actor's current goals* $\mathcal{GS} \in \mathcal{VS}(\mathcal{A})$.

Proof sketch: *Let* $\mathcal{GS} = g_1, ..., g_n$ *be the actor's goals. By assumption A1, the observed actions* $\mathcal{A}$ *will be the partial, interleaved execution of plans* $p_1, ..., p_n$ *where the actor executes plan* p_i *for goal* g_i*. By assump-*

[1] The terms in this definition are defined formally in [Lesh and Etzioni, 1995].

tion A2, since the actor executes plan p_i for goal g_i, $\langle p_i, g_i \rangle \in \mathcal{PL}$. By the definition of consistency, these two statements entail that $\mathcal{GS}$ will be consistent with $\mathcal{A}$. By assumptions A2 and A3 and the definition of version spaces, $\mathcal{GS} \in \mathcal{VS}(\emptyset)$. By the definition of version spaces, since $\mathcal{GS} \in \mathcal{VS}(\emptyset)$ and $\mathcal{GS}$ is consistent with $\mathcal{A}$, $\mathcal{GS} \in \mathcal{VS}(\mathcal{A})$.

We order goals by the generality relationship. If every plan for g_i is a plan or subplan for g_j, then g_j is *more general than* g_i, written $g_i \preceq g_j$. For example, "mow the lawn" $\preceq$ "mow the lawn and pay the bills" and "capture Bad Bill, alive" $\preceq$ "capture Bad Bill". Formally, $g_i \preceq g_j$ if every action sequence consistent with g_i is also consistent with g_j. If $g_i \preceq g_j$ and $g_j \not\preceq g_i$ then $g_i \prec g_j$. This closely resembles the definition of generality between plans in [Weida and Litman, 1992].

We represent $\mathcal{VS}(\mathcal{A})$ with a three-tuple $\langle \mathcal{S}, \mathcal{G}, \mathcal{A} \rangle$, where $\mathcal{S}$ is the set of most specific, consistent goals and $\mathcal{G}$ is the set of most general, consistent goals. A goal g_1 is *between* goals g_0 and g_2 iff $g_0 \prec g_1 \prec g_2$. The pair $\langle \mathcal{S}, \mathcal{G} \rangle$ *bounds* $\mathcal{VS}$ iff for every goal $g_i \in \mathcal{VS}$, either g_i is in $\mathcal{S}$ or $\mathcal{G}$ or there is a goal $g_s \in \mathcal{S}$ and goal $g_g \in \mathcal{G}$ such that g_i is between g_s and g_g.

We use consistency graphs [Lesh and Etzioni, 1995] to represent the consistent plans for each goal in $\mathcal{S}$ and $\mathcal{G}$. The RIGS algorithm takes a goal set $\mathcal{GS}$, Λ, and $\mathcal{A}$, and constructs and manipulates a consistency graph in order to identify which goals in $\mathcal{GS}$ are inconsistent with $\mathcal{A}$. RIGS runs in time that is linear in $|\mathcal{GS}|$ and polynomial in $|\Lambda| + |\mathcal{A}|$.

6 Goal recognition algorithm

We now present the BOCE goal recognition algorithm. First we describe a bias-independent algorithm that calls four bias-specific functions. Then we describe the bias-specific functions that BOCE requires for two classes of conjunctive search goals. Finally, we extend BOCE to handle multiple goals and interleaved plans.

6.1 Boce algorithm

The Based-on-Candidate-Elimination (BOCE) goal recognition algorithm, shown in figure 2, closely resembles the Candidate Elimination algorithm's [Mitchell, 1982] update procedure for a positive example. The pseudo code in figure 2 can be viewed as a template for an algorithm. One needs to "plug in" the *Init*, *Compare*, *Consistent*, and *MinGen* functions (see table 1) to instantiate the goal recognizer. These functions depend on the plan and goal bias. The *Consistent* function for the directed-weak plan bias is described in [Lesh and Etzioni, 1995]. In section 6.2, we describe the *Init*, *Compare*, and *MinGen* for two classes of conjunctive search goals.

The following is a brief sketch of how the algorithm works. The initial library is set to $\langle \mathcal{S}_0, \mathcal{G}_0, \mathcal{A}_0 = \emptyset \rangle$ where $\mathcal{S}_0$ and $\mathcal{G}_0$ are the sets returned by *Init*. The update procedure works as follows. Suppose the current library is $\langle \mathcal{S}_n, \mathcal{G}_n, \mathcal{A}_n \rangle$ and we observe action a_{n+1}. First, BOCE adds a_{n+1} to the end of sequence $\mathcal{A}_n$ to produce $\mathcal{A}_{n+1}$. Then BOCE uses *Consistent* to determine which goals in $\mathcal{S}_n$ and $\mathcal{G}_n$ are inconsistent with $\mathcal{A}_{n+1}$. BOCE removes inconsistent goals from $\mathcal{G}_n$ to produce $\mathcal{G}_{n+1}$ and uses *MinGen* to replace inconsistent goals in $\mathcal{S}_n$ with their minimal generalizations to produce $\mathcal{S}_{n+1}$. BOCE uses *Compare* to remove goals from $\mathcal{S}_{n+1}$ that are more general than other goals in $\mathcal{S}_{n+1}$ or less general than any goal removed from $\mathcal{G}_n$.

- BOCE-initialize(Ω)
  1. Let $\langle \mathcal{S}_0, \mathcal{G}_0 \rangle = \mathit{Init}(\Omega)$
  2. Let $\mathcal{A}_0 = \emptyset$
- BOCE-observe(a_n, Λ, Ω)
  1. Let $A_n = \mathcal{A}_{n-1} + a_n$ (the sequence A_{n-1} with a_n appended to the end)
  2. Let $\langle \mathcal{S}_n, \mathcal{G}_n \rangle = \langle \mathcal{S}_{n-1}, \mathcal{G}_{n-1} \rangle$
  3. FOR every goal $g_i \in \mathcal{G}_n$
     - IF *Consistent*$(g_i, \mathcal{A}_n)$ is `false` THEN remove g_i from $\mathcal{G}_n$
  4. FOR every goal $s_i \in \mathcal{S}_n$
     - IF *Consistent*$(s_i, \mathcal{A}_n)$ is `false` THEN
       (a) remove s_i from $\mathcal{S}_n$
       (b) add to $\mathcal{S}_n$ all goals m_i in *MinGen*$(s_i, \mathcal{A}_n)$ such that *Consistent*$(s_i, \mathcal{A}_n)$ is `true` and *Compare*(s_i, g_j) is `true` for some $g_j \in \mathcal{G}_n$
- BOCE(Ω, Λ)
  1. BOCE-initialize(Ω)
  2. As each action a_n is observed
     (a) BOCE-observe(a_n, Λ, Ω)
     (b) IF $\mathcal{G}_n = \mathcal{S}_n = \{g_i\}$ then return Nec(g_i) $\wedge$ Suff(g_i)
     (c) ELSE IF $\mathcal{S}_n = \{g_i\}$ and `PartialSuff` THEN return Suff(g_i)
     (d) ELSE IF $\mathcal{S}_n = \{g_i\}$ and `PartialNec` THEN return Nec(g_i)
     (e) ELSE IF $\mathcal{S}_n$ or $\mathcal{G}_n$ is empty THEN return `fail`
     (f) ELSE return `null` (*indicating that more observations are needed to recognize goal*)

Figure 2: Pseudo code for BOCE

The $\mathcal{S}$ and $\mathcal{G}$ boundaries produced by BOCE always describe the version space of consistent goals.

Theorem 1 (Correct boundaries)
Let $\langle \mathcal{S}_n, \mathcal{G}_n, \mathcal{A}_n \rangle$ be the library that BOCE produces after observing actions $a_1, a_2, ..., a_n$. Pair $\langle \mathcal{S}_n, \mathcal{G}_n \rangle$ bounds $\mathcal{VS}(a_1, a_2, ..., a_n)$.

Proof sketch: *This is a proof sketch by induction. The base case is that $\langle \mathcal{S}_0, \mathcal{G}_0 \rangle$ bounds $\mathcal{VS}(\emptyset)$.*

The induction step is that if $\langle \mathcal{S}_{n-1}, \mathcal{G}_{n-1} \rangle$ *bounds* $\mathcal{VS}(a_1, ..., a_{n-1})$ *then* $\langle \mathcal{S}_n, \mathcal{G}_n \rangle$ *bounds* $\mathcal{VS}(a_1, ..., a_n)$.

The base case holds because BOCE initializes the boundaries by calling the Init *function. As defined in table 1,* Init *returns a pair that bounds* $\mathcal{VS}(\emptyset)$.

By definition, $\mathcal{VS}(a_1 ... a_{n-1}) \subseteq \mathcal{VS}(a_1 ... a_n)$. *Thus, given that* $\langle \mathcal{S}_{n-1}, \mathcal{G}_{n-1} \rangle$ *bounds* $\mathcal{VS}(a_1, ..., a_{n-1})$, *Step 2 of BOCE-observe establishes that* $\langle \mathcal{S}_n, \mathcal{G}_n \rangle$ *bounds* $\mathcal{VS}(a_1, ..., a_n)$. *We now show that steps 3 and 4 of BOCE-observe preserve this relationship. Step 3 removes inconsistent goals from* $\mathcal{G}_n$. *Removing a goal* g_g *from* $\mathcal{G}$ *can only remove goals that are less general than* g_g *from the space bounded by* $\langle \mathcal{S}_n, \mathcal{G}_n \rangle$. *By the definition of generality, any goal that is less general than an inconsistent goal is, itself, inconsistent. Any goal* g_i *removed from* $\mathcal{S}_n$ *by step 4a is immediately replaced in step 4b by its minimal generalizations. Thus the only goal removed from the space bounded by* $\langle \mathcal{S}_n, \mathcal{G}_n \rangle$ *by step 4 is* g_i *itself, which must be inconsistent or else step 4 would not remove it. Thus, steps 3 and 4 only remove inconsistent goals from the space bounded by* $\langle \mathcal{S}_n, \mathcal{G}_n \rangle$. *Since* $\mathcal{VS}(\mathcal{A})$ *contains only consistent goals,* $\langle \mathcal{S}_n, \mathcal{G}_n \rangle$ *bounds* $\mathcal{VS}(\mathcal{A})$.

Once the $\mathcal{S}$ and $\mathcal{G}$ boundaries are computed, we can use them to perform both full and partial recognition. As shown in step 2b of the BOCE function in figure 2, BOCE reports g is both sufficient and necessary for the actor's goals if $\mathcal{S}$ and $\mathcal{G}$ ever converge to g.

Theorem 2 (Full recognition) *If pair* $\langle \{g\}, \{g\} \rangle$ *bounds* $\mathcal{VS}(\mathcal{A})$ *then* g *is necessary and sufficient.*

Proof sketch: *Since* $\langle \{g\}, \{g\} \rangle$ *bounds* $\mathcal{VS}(\mathcal{A})$, *then* $\mathcal{VS}(\mathcal{A}) = \{g\}$. *Recall that* g *is actually a combination of goals. By lemma 1,* $\mathcal{GS} \in \mathcal{VS}(\mathcal{A})$. *Thus* $g = \mathcal{GS}$. *Thus each goal in* g *is both necessary and sufficient, by the definition of necessary and sufficient goals.*

Although we do not formalize it here, it is also possible to recognize one of the actor's goals without recognizing all of the actor's goals. If, for example, $\langle \{g_1 \wedge g_2\}, \{g_1 \wedge g_3\} \rangle$ bounds the version space then g_1 must be one of the actor's goals.

The actor's goals can be partially described if generality between goals has a fixed correlation to entailment between goals. The boolean `PartialSuff` is set to true only if for all goals g_i, g_j in $\mathcal{PL}$, g_i entails g_j if $g_i \preceq g_j$. Below, we define a set of goals for which this is true. If `PartialSuff` is true, then in step 2c BOCE indicates that goal g_i is sufficient if the $\mathcal{S}$ boundary ever converges to g_i. Similarly, the boolean `PartialNec` is set to true only if for all goals g_i, g_j in $\mathcal{PL}$, g_j entails g_i if $g_i \preceq g_j$. And if `PartialNec` is true then in step 2d BOCE outputs that goal g_i is necessary for one of the actor's goals if the $\mathcal{S}$ boundary ever converges to g_i.

Theorem 3 (Partial recognition) *If pair* $\langle \{g\}, \mathcal{G} \rangle$ *bounds* $\mathcal{VS}(\mathcal{A})$ *and* $g_i \prec g_j$ *implies* g_i *entails* g_j, *then* g *is sufficient. If* $\langle \{g\}, \mathcal{G} \rangle$ *bounds* $\mathcal{VS}(\mathcal{A})$ $g_i \prec g_j$ *implies* g_j *entails* g_i, *then* g *is necessary.*

function	definition
Init(Ω)	$\langle \mathcal{S}, \mathcal{G} \rangle$ that bounds $\mathcal{VS}(\emptyset)$
Consistent($g, \mathcal{A}, \Lambda$)	*true* iff g is consistent with $\mathcal{A}$
Compare(g_i, g_j)	*true* iff $g_i \prec g_j$
MinGen($g, \mathcal{A}$)	$\mathcal{GS}$ st. $\forall g_i \in \mathcal{VS}(\mathcal{A})$, $(g \not\prec g_i) \vee (\exists g_j (g_j \in \mathcal{GS}) \wedge (g_j \preceq g_i))$

Table 1: BOCE's bias-specific functions.

Proof sketch: *Since* $\langle \{g\}, \mathcal{G} \rangle$ *bounds* $\mathcal{VS}(\mathcal{A})$ $\{g\}$ *is more specific than every other goal in* $\mathcal{VS}(\mathcal{A})$. *Thus, by lemma 1,* $\{g\} \preceq \mathcal{GS}$ *where* $\mathcal{GS}$ *is the actor's goal. If* $\{g\}$ *is more specific than the actor's goal and* $g_i \prec g_j$ *implies* g_j *entails* g_j, *then* $\mathcal{GS}$ *must entail* $\{g\}$. *By the definition of necessary, if the actor's goal* $\mathcal{GS}$ *entails* $\{g\}$, *then* $\{g\}$ *is necessary. Similarly, if* $\{g\}$ *is more specific than the actor's goal, and* $g_i \prec g_j$ *implies* g_i *entails* g_j, *then* $\{g\}$ *must entail the actor's goal. By the definition of sufficiency, if* $\{g\}$ *entails* $\mathcal{GS}$ *then* g *is sufficient.*

If $\mathcal{S} = \emptyset$ or $\mathcal{G} = \emptyset$ then one of BOCE's assumptions has been violated, and BOCE returns `fail` at step 2e.

Theorem 4 (Failure) *If* $\langle \mathcal{S}, \emptyset \rangle$ *or* $\langle \emptyset, \mathcal{G} \rangle$ *bounds* $\mathcal{VS}(\mathcal{A})$ *then assumptions 1, 2, 3, and 4 cannot all hold.*

Proof sketch: *If* $\mathcal{S} = \emptyset$ *or* $\mathcal{G} = \emptyset$ *then, by the definition of bounds,* $\mathcal{VS}(\mathcal{A}) = \emptyset$. *By lemma 1,* $\mathcal{GS} \in \mathcal{VS}(\mathcal{A})$ *and thus* $\mathcal{VS}(\mathcal{A}) \neq \emptyset$. *Since lemma 1 follows from assumptions 1, 2, 3, and 4 if* $\mathcal{S}$ *or* $\mathcal{G}$ *is empty, then all of our assumptions cannot hold.*

Finally, in step 2f BOCE outputs `null` if none of the above recognition conditions occur. The output `null` simply means "no comment at this time" which is always true. Thus, the above theorems show that BOCE's outputs are always true.

6.2 Goal bias specific functions

In this section, we discuss two classes of goals: decomposable and composable goals. Decomposable and composable goal sets are mutually exclusive but not exhaustive (some goal sets are neither composable or decomposable). Table 2 describes the *Init*, *Compare*, and *MinGen* functions for these two classes of goals.

We discuss decomposable goals first. Goals are decomposable if adding a conjunct makes them more specific. For example, adding the constraint that the printer can print color to the goal of finding a free printer forms a more specific goal. Formally, goal set $\mathcal{GS}$ is

decomposable if for all $g_i, g_j \in \mathcal{GS}$, $g_j \subset g_i$ iff $g_i \prec g_j$.[2]

In table 2, we represent sets of goals with a set of literals $\mathcal{I}$ and a set of literals $\mathcal{C}$. The literals in $\mathcal{I}$ describe the invariant part of the goals that appears in every goal. For example, in the set of goals of looking for a free printer with some subset of possible printer attributes, the invariant part is "the printer must be free". The set $\mathcal{C}$ describes possible constraints on the goal, such as "the printer must print color". We use $\mathcal{I} \times \mathcal{C}^*$ to refer to the set of goals $\{\mathcal{I} \cup \mathcal{C}_i \mid \mathcal{C}_i \subseteq \mathcal{C}\}$.

By the definition of the decomposable goal sets, the *Compare* function for decomposable goals is, as shown in table 2, the inverse of the subset function. The *Init* function for decomposable goal set $\mathcal{I} \times \mathcal{C}^*$ is $\langle\{\mathcal{I} \cup \mathcal{C}\}, \{\mathcal{I}\}\rangle$. The $\mathcal{S}$ set contains the most constrained and thus the most specific goal. The $\mathcal{G}$ set contains the least constrained and thus most general goal. This definition of *Init* satisfies BOCE's requirements for the *Init* function, as specified in table 1.

Theorem 5 (Init Decomposable) *If $\mathcal{I} \times \mathcal{C}^*$ is decomposable, then $\langle\{\mathcal{I} \cup \mathcal{C}\}, \{\mathcal{I}\}\rangle$ bounds $\mathcal{I} \times \mathcal{C}^*$.*

Proof: *Let g_i be a goal in $\mathcal{I} \times \mathcal{C}^*$. Goal g must be of the form $\mathcal{I} \cup \mathcal{C}_j$ for some $\mathcal{C}_j \subseteq \mathcal{C}$. By the definition of decomposable goal sets, since $\mathcal{I} \subseteq (\mathcal{I} \cup \mathcal{C}_j) \subseteq (\mathcal{I} \cup \mathcal{C})$, $(\mathcal{I} \cup \mathcal{C}) \preceq (\mathcal{I} \cup \mathcal{C}_j) \preceq \mathcal{I}$. By the definition of bounds, since every goal g_i in $\mathcal{I} \times \mathcal{C}^*$ is between $\mathcal{I} \cup \mathcal{C}$ and $\mathcal{I}$, $\langle\{\mathcal{I} \cup \mathcal{C}\}, \{\mathcal{I}\}\rangle$ bounds $\mathcal{I} \times \mathcal{C}^*$.*

We compute the minimal generalizations of goal $\mathcal{I} \cup \mathcal{C}_i$ as follows. For all $c_j \in \mathcal{C}_i$, we call *Consistent* to determine if $\mathcal{I} + c_j$ is consistent with $\mathcal{A}$. If $\mathcal{I} + c_j$ is inconsistent, we remove it from $\mathcal{C}_i$. The resulting goal, $\mathcal{I} \cup \mathcal{C}'_i$, is the sole generalization of $\mathcal{I} \cup \mathcal{C}_i$. This definition of MinGen, shown in table 2, satisfies BOCE's requirements for the MinGen function, shown in table 1.

Theorem 6 (MinGen Decomposable) *If $\mathcal{I} \times \mathcal{C}^*$ is decomposable and $\mathcal{C}_i \subseteq \mathcal{C}$, then the minimal generalizations of $\mathcal{I} \cup \mathcal{C}_i$ in $\mathcal{I} \times \mathcal{C}^*$ is $\{\mathcal{I} \cup \mathcal{C}'_i\}$ where $\mathcal{C}'_i$ is the set of all literals c_j in $\mathcal{C}_i$ such that $\mathcal{I} + c_j$ is consistent with the observed actions $\mathcal{A}$.*

Proof sketch: *To show $\{\mathcal{I} \cup \mathcal{C}'_i\}$ is the minimal generalizations of $\mathcal{I} \cup \mathcal{C}_i$, we show that for all goals $g_k \in \mathcal{VS}(\mathcal{A})$ either* [1] *$(\mathcal{I} \cup \mathcal{C}_i) \not\preceq g_k$ or* [2] *$(\mathcal{I} \cup \mathcal{C}'_i) \preceq g_k$.*

The version space $\mathcal{VS}(\mathcal{A})$ contains all goals in $\mathcal{I} \times \mathcal{C}^*$ that are consistent with $\mathcal{A}$. Since $g_k \in \mathcal{I} \times \mathcal{C}^*$, $g_k = \mathcal{I} \cup \mathcal{C}_l$ for some $\mathcal{C}_l \subseteq \mathcal{C}$. Note that $\mathcal{C}'_i \subseteq \mathcal{C}_i \subseteq \mathcal{C}$. By the definition of decomposable goals, if $\mathcal{C}_l \subseteq \mathcal{C}'_i$ then $(\mathcal{I} \cup \mathcal{C}'_i) \preceq g_k$ and condition [2] *is met. By the definition of decomposable goals, if $\mathcal{C}_l \not\subseteq \mathcal{C}_i$ then $(\mathcal{I} \cup \mathcal{C}_i) \not\preceq g_k$ and condition* [1] *is met. If neither $\mathcal{C}_l \subseteq \mathcal{C}'_i$ nor $\mathcal{C}_l \not\subseteq \mathcal{C}_i$, then there exists some c_m such that $(c_m \in \mathcal{C}_l) \wedge (c_m \in \mathcal{C}_i) \wedge (c_m \notin \mathcal{C}'_i)$. By the way $\mathcal{C}'_i$ is constructed, if c_m is in $\mathcal{C}_i$ but not $\mathcal{C}'_i$ then $(\mathcal{I} + c_m)$ must be inconsistent with the observed actions. Since $c_m \in \mathcal{C}_l$, $(\mathcal{I} + c_m) \subseteq g_k$. By the definition of decomposable goals, $g_k \preceq (\mathcal{I} + c_m)$. By the definition of generality, if $g_k \preceq (\mathcal{I} + c_m)$ and $(\mathcal{I} + c_m)$ is inconsistent, then g_k is also inconsistent. Thus $g_k \notin \mathcal{VS}(\mathcal{A})$. Thus, for all $g_k \in \mathcal{VS}(\mathcal{A})$ conditions 1 or 2 above is met.*

[2] A conjunctive goal g_i is a subset of g_j iff every conjunct of g_i is also a conjunct of g_j.

function	decomposable goal set $\mathcal{I} \times \mathcal{C}^*$	composable goal set $\mathcal{I} \times \mathcal{C}^*$
Init()	$\langle\{\mathcal{I} \cup \mathcal{C}\}, \{\mathcal{I}\}\rangle$	$\langle\{\mathcal{I}\}, \{\mathcal{I} \cup \mathcal{C}\}\rangle$
Compare(g_i, g_j)	*true* iff $g_i \supset g_j$	*true* iff $g_i \subset g_j$
MinGen(g)	Let $g = \mathcal{I} \cup \mathcal{C}_i$. Return $\{\mathcal{I} \cup \mathcal{C}'_i\}$ s.t. $\forall c_j \in \mathcal{C}_i$, Cons($\mathcal{I}$+$c_j, \mathcal{A}$)	Let $g = \mathcal{I} \cup \mathcal{C}_i$. Return $\{\mathcal{I} \cup \mathcal{C}_i + c_j \mid c_j \in (\mathcal{C} - \mathcal{C}_i)\}$
`PartialSuff`	True	False
`PartialNec`	False	True

Table 2: Bias-specific functions for composable and decomposable goals

BOCE processes exponentially large decomposable goals in linear time. This is possible because the $\mathcal{S}$ and $\mathcal{G}$ sets always contain at most one goal. Both sets start out as singletons. In general, BOCE can increase the size of $\mathcal{S}$ by replacing a goal with its minimal generalizations. The minimal generalizations of a decomposable goal, however, is always a set containing exactly one goal. Furthermore, the minimal generalizations of a decomposable goal can be computed by checking the consistency of at most $|\mathcal{C}|$ goals. The total number of consistency checks for a single observed actions is, then, $|\mathcal{C}| + 2$ including the two checks of the goals in $\mathcal{S}$ and $\mathcal{G}$. This is logarithmic time because $\mathcal{I} \times \mathcal{C}^*$ contains $2^{|\mathcal{C}|}$ goals.

Theorem 7 (Logarithmic time decomposable) *If the goals in $\mathcal{PL}$ are the decomposable goal set $\mathcal{I} \times \mathcal{C}^*$, then BOCE will compute $\langle\mathcal{S}_{n+1}, \mathcal{G}_{n+1}, \mathcal{A}_{n+1}\rangle$ given $\langle\mathcal{S}_n, \mathcal{G}_n, \mathcal{A}_n\rangle$ and action a_{n+1} by calling the consistency function on at most $|\mathcal{C}| + 2$ goals.*

Proof sketch: *As shown in table 2, for decomposable goal sets, $|\mathcal{G}_0|$ and $|\mathcal{S}_0| = 1$. Since BOCE never adds to the $\mathcal{G}$ set, for all n, $|\mathcal{G}_n| \leq 1$. BOCE can only increase $|\mathcal{S}|$ by replacing a goal in $\mathcal{S}$ with its minimal generalizations. Since, as shown in table 2, the minimal generalizations of any decomposable goal is always a single goal, for all n, $|\mathcal{S}_n| \leq 1$.*

BOCE checks the consistency of goals only in steps 3 and 4 of BOCE-update. In step 3, BOCE-update checks the consistency of $|\mathcal{G}_n| = 1$ goal. In step 4, BOCE-update checks the consistency of $|\mathcal{S}_n| = 1$ goal and calls MinGen *on every inconsistent goal. The MinGen($\mathcal{I} \cup \mathcal{C}_i$) needs to determine if goal $(\mathcal{I} + c_j)$ is consistent for all $c_j \in \mathcal{C}_i$. This requires exactly $|\mathcal{C}_i|$ calls to the consistency function. Note that $\mathcal{C}_i \subset \mathcal{C}$.*

Thus the total calls in steps 3 and 4 of BOCE-update is at most $|\mathcal{C}|+2$. No where else does BOCE check the consistency of a goal.

The `PartialSuff` flag can be set to `true` for decomposable goal sets, indicating that if $g_i \prec g_j$ then g_i entails g_j. This is true because if g_i and g_j are decomposable and if $g_i \prec g_j$ then $g_j \subset g_i$; if $g_j \subset g_i$ then g_j holds for all worlds in which g_j is true and thus g_i entails g_j. For example, consider the goal of finding a free, color, second-floor printer, or $g_{fc2} = (\text{ISA printer } x) \wedge (\text{FREE } x) \wedge (\text{COLOR } x) \wedge (\text{FLOOR } x\ 2)$. A subset of this goal is $g_{fc} = (\text{ISA printer } x) \wedge (\text{FREE } x) \wedge (\text{COLOR } x)$, the goal of finding a free, color printer. Goal g_{fc2} is sufficient for goal g_{fc}, and will be sufficient for any of its subsets.

Now we discuss composable goals. Goals are composable if adding a conjunct makes them more general. For example, if A and B are independent conjuncts then "do A" $\prec$ "do A and B" because anything we observe in service of "do A" could also be in service of "do A and B". Formally, goal set $\mathcal{G}$ is *composable* if for all $g_i, g_j \in \mathcal{G}$, $g_i \subset g_j$ iff $g_i \prec g_j$.

By the definition of the composable goal sets, the *Compare* function for decomposable goals is, as shown in table 2, the subset function. The *Init* function for decomposable goal set $\mathcal{I} \times \mathcal{C}^*$ is $\langle \{\mathcal{I}\}, \{\mathcal{I} \cup \mathcal{C}\} \rangle$. This definition of *Init* satisfies BOCE's requirements for the *Init* function, as specified in table 1.

Theorem 8 (Init Composable) *If $\mathcal{I} \times \mathcal{C}^*$ is composable, then $\langle \{\mathcal{I}\}, \{\mathcal{I} \cup \mathcal{C}\} \rangle$ bounds $\mathcal{I} \times \mathcal{C}^*$.*

The proof is very similar to the proof of theorem 5. For composable goal sets, adding a constraint makes the goal more general. Thus, the minimal generalizations of a composable goal are simply all those goals with exactly one more constraint. Formally,

Theorem 9 (MinGen Composable) *If $\mathcal{I} \times \mathcal{C}^*$ is decomposable and $\mathcal{C}_i \subseteq \mathcal{C}$, then the minimal generalizations of $\mathcal{I} \cup \mathcal{C}_i$ in $\mathcal{I} \times \mathcal{C}^*$ is $\{\mathcal{I} \cup \mathcal{C}_i + c_j \mid c_j \in (\mathcal{C} - \mathcal{C}_i)\}$*

Proof sketch: *Let $g_k \in \mathcal{VS}(\mathcal{A})$. The version space $\mathcal{VS}(\mathcal{A})$ contains all goals in $\mathcal{I} \times \mathcal{C}^*$ that are consistent with $\mathcal{A}$. Since $g_k \in \mathcal{I} \times \mathcal{C}^*$, $g_k = \mathcal{I} \cup \mathcal{C}_l$ for some $\mathcal{C}_l \subseteq \mathcal{C}$. If $\mathcal{C}_i \not\subset \mathcal{C}_l$, then by the definition of composable goals $(\mathcal{I} \cup \mathcal{C}_i) \not\prec g_k$. If $\mathcal{C}_i \subset \mathcal{C}_l$, then there must be some goal $g_j \in \mathcal{MG} = \{\mathcal{I} \cup \mathcal{C}_i + c_j \mid c_j \in (\mathcal{C} - \mathcal{C}_i)\}$ such that $g_j \subseteq g_k$. By the definition of composable goals, $g_j \preceq g_k$. Thus, since either $\mathcal{C}_i \not\subset \mathcal{C}_l$ or $\mathcal{C}_i \subset \mathcal{C}_l$, for all $g_k \in \mathcal{VS}(\mathcal{A})$, either $(\mathcal{I} \cup \mathcal{C}_i) \not\prec g_k$ or $\exists g_j (g_j \in \mathcal{MG}) \wedge (g_j \prec g_k)$. Thus, by the definition in table 1, $\mathcal{MG}$ is the minimal generalizations of $(\mathcal{I} \cup \mathcal{C}_i)$.*

We cannot process composable goal sets as efficiently as decomposable ones. In the worst case, *MinGen*(g) produces $|\Omega|$ goals and thus the $\mathcal{S}$ and $\mathcal{G}$ boundaries can grow to size $2^{|\Omega|-1}$, half the total number of goals. In section 9 we show that, in practice, the boundaries for composable goals remain small.

The `PartialNec` flag can be set to `true` for composable goal sets, indicating that if $g_i \prec g_j$ then g_j entails g_i. This is true because if g_i and g_j are composable and if $g_i \prec g_j$ then $g_i \subset g_j$; if $g_i \subset g_j$ then g_i holds for all worlds in which g_j holds and thus g_j entails g_i.

Finally, note that our approach is modular. From the definition of bounds, it follows that if $\langle \mathcal{S}_1, \mathcal{G}_1 \rangle$ bounds version space $\mathcal{VS}_1$ and $\langle \mathcal{S}_2, \mathcal{G}_2 \rangle$ bounds $\mathcal{VS}_2$ then $\langle \mathcal{S}_1 \cup \mathcal{S}_2, \mathcal{G}_1 \cup \mathcal{G}_2 \rangle$ bounds version space $\mathcal{VS}_1 \cup \mathcal{VS}_2$. Thus, we can run BOCE separately with different sets of bias-specific functions for different goal sets and easily merge the results to produce the boundaries of the union of the goal sets.

6.3 Interleaved plans

We now show how to perform recognition on interleaved plans, given the ability to perform recognition on uninterleaved plans. Suppose we have *Consistent*, *Compare*, *Init*, and *MinGen* for $K = 1$. The objective is construct *Consistent*^{K}, *Compare*^{K}, *Init*^{K}, and *MinGen*^{K} for $K > 1$. Recall that we equate having goals g_1 and g_2 with having goal $g_1 \wedge g_2$. A simple but slow approach would simply be to represent all combinations of goals explicitly, but we show that version spaces allow for more tractable representation.

The function *Consistent*^{K} is the same as *Consistent* because $g_1 \wedge ... \wedge g_n$ is a goal just like any other goal. In general, it is not possible to construct *Compare*^{K} because it depends on the plan bias. For example, if the plan bias states that people execute optimal plans then the set of plans for goal g_1 and the set of plans for goal g_2 might have nothing in common with the set of plans for $g_1 \wedge g_2$. We make the following assumption, which holds for our directed-weak plan bias:

A5. Let $s_1, ..s_n$ and $g_1, ..., g_m$ all be goals. $(s_1 \wedge s_2 \wedge ..s_n) \preceq (g_1 \wedge g_2 \wedge ..g_m)$ iff there is an onto mapping f such that for all $1 \leq i \leq n$, $s_i \preceq g_{f(i)}$.

Thus, for example, $g_1 \preceq (g_1 \wedge g_2)$ and if $g_1 \preceq g_3$ and $g_2 \preceq g_4$ then $(g_1 \wedge g_2) \preceq (g_3 \wedge g_4)$. Under this assumption, interleaved goals behave very much like composable goals. The *Init*^{K} function returns $\langle \mathcal{S}, \mathcal{G}^K \rangle$. It follows from assumption A5 that if $\langle \mathcal{S}, \mathcal{G} \rangle$ bounds $\mathcal{VS}(\mathcal{A})$ then $\langle \mathcal{S}, \mathcal{G}^K \rangle$ bounds $\mathcal{VS}(\mathcal{A})^K$.

Version spaces are useful for handling interleaved goals because the *MinGen*^{K} function can exploit a rich generality structure amongst combinations of goals. Let $\mathcal{MG}^1$ be the minimal generalizations of goal g_1 and $\mathcal{MG}^2$ be the minimal generalizations of goal g_2. It is true that $\mathcal{MG}^1 \times \mathcal{MG}^2$ is a minimal generalization of $g_1 \wedge g_2$. However, there is a more compact, minimal generalization of $g_1 \wedge g_2$, defined as follows:

Theorem 10 (MinGen^K) *The minimal generalizations of goal $g_1 \wedge g_2 \wedge ..g_n$ in $\mathcal{GS}^K$ are all goals of the form $MGJ_i \wedge g_1 \wedge ...g_{j-1} \wedge g_{j+1} \wedge ..g_n$, where $MGJ_i \in$ MinGen(g_j) and, if $n < K$, goals of the form $g_1 \wedge ... \wedge g_n \wedge g_{n+1}$ where $g_{n+1} \in \mathcal{S}_0$ where $\langle \mathcal{S}_0, \mathcal{G}_0 \rangle$ is the pair returned by* Init *that bounds $\mathcal{GS}$.*

Due to space limitations, we omit the proof. If every goal in $\mathcal{GS}$ has Q minimal generalizations then a goal in $\mathcal{GS}^K$ has at most $(Q \times K) + |\mathcal{S}|$ minimal generalizations. This is much better than the Q^K minimal generalizations that occurs from taking the cross product of the individual generalizations of all the goals.

7 Example

The following example illustrates our notion of biases, the I/O of goal recognition, and how our algorithm works. We begin with an informal description of the interactions among a computer user, a task-completion agent, and a goal recognizer. The user's goal is to find a free, color printer.

First, the user checks printer `alpha`, a color, fourth-floor, double-sided printer made by Xerox in 1991. The goal recognizer ignores where and when the printer was made and tells the task-completion agent that finding a free, color, fourth-floor, double-sided printer is *sufficient* for the actor's goal. The agent, however, fails to locate a printer meeting these requirements.

Next, the user checks printer `beta`, a color, second-floor, double-sided printer made by Xerox in 1993. Now, the recognizer reports that a free, color, double-sided printer is sufficient (dropping the fourth-floor requirement). The task-completion agent finds printer `gamma`, a color, third-floor, double-sided printer made by NEC in 1991. The agent recommends `gamma` to the user, which satisfies her goal.

The biases in section 4 allow BOCE to draw these conclusions. Without a conjunctive goal bias it could not exclude, for example, the possibility that the actor was searching for a printer on the second *or* fourth floor, in which case printer `gamma` would be unsuitable. Without a plan bias that excludes irrelevant actions, the recognizer could not even infer that the actor's goals had anything to do with finding a printer.

For the above interactions to occur, the predicate set Ω must include COLOR, FLOOR, and DOUBLE.SIDED. But Ω must exclude YEAR.MADE and MADE.BY. Otherwise, the recognizer might have instructed the task-completion agent to find a printer made by Xerox. Finally, K must be 1. Otherwise, the two observed actions could have been for two separate goals.

We now describe the $\mathcal{S}$ and $\mathcal{G}$ boundaries that BOCE maintains, assuming the printer search goals are decomposable. $\mathcal{G}_0 = \{(\text{ISA printer } x) \wedge (\text{FREE } x)\}$, i.e. it contains the goal of finding a free printer. $\mathcal{S}_0 = \{(\text{ISA printer } x) \wedge (\text{FREE } x) \wedge (\text{FLOOR } x\ F) \wedge (\text{DOUBLE.SIDED } x) \wedge (\text{COLOR } x)\}$. The variable F is a special *schema* variable, which represents all constants appropriate for the FLOOR predicate. It is reasonable to view the goal in $\mathcal{S}_0$ as requiring that the printer be on the first floor, on the second floor, on the third floor, etc. After the computer user polls printer `alpha`, $\mathcal{S}_1$ contains $\{(\text{ISA printer } x) \wedge (\text{FREE } x) \wedge (\text{FLOOR } x\ 4) \wedge (\text{DOUBLE.SIDED } x) \wedge (\text{COLOR } x)\}$. The $\mathcal{G}$ set is unchanged; $\mathcal{G}_1 = \mathcal{G}_0$. After the computer user polls printer `beta`, $\mathcal{S}_2$ contains $\{(\text{ISA printer } x) \wedge (\text{FREE } x) \wedge (\text{DOUBLE.SIDED } x) \wedge (\text{COLOR } x)\}$. Again, the $\mathcal{G}$ set is unchanged. At each step, the goal in $\mathcal{S}$ is reported to be sufficient for the actor's goal.

8 Discussion of scope

The simplicity of the above example, designed to illustrate our approach, does not reflect the full scope of our work. As the example suggests, our approach is limited to *keyhole* recognition where the observed actor is unaware of or indifferent to the observer. The example should not be construed to suggest we allow only 'single-level" plans in which every action contributes directly to the goal. Our formulation and the RIGS algorithm allow for arbitrarily "deep" plans with subgoals, subgoals of subgoals, and so on.

The above example is also small. Section 9 describes experiments on large libraries in the Unix and Toast [Agre and Horswill, 1992] domains. We have recently tested BOCE, with similar results, on data gathered under Microsoft Windows 95.[3] Since user help systems are a primary application of keyhole recognition, software systems are an important domain. Additionally, conjunctive search goals are likely to occur in any domains in which people search for objects. Furthermore, our approach could easily be extended to *conjunctive set goals*, such as the goal of compressing all files that are large and have been unmodified for a year. Conjunctive set and search goals are important because they give rise to long plans and thus task-completion can be especially helpful. Additionally, our experiments in the Toast domain show that our recognizer can work on reactive agents that do not actually construct plans, but do act planfully.

What about classes of goals other than composable and decomposable goals? To review, BOCE exploits the generality relationship among goals to maintain a compact representation of the candidate goals. Broadly speaking, this approach will work well when (1) the library has a large number of goals, (2) there is a rich generality structure among these goals, and (3) this generality structure can be determined by the form, rather than the content, of the goals. A case

[3] Contact neal@cs.washington.edu for details and data.

predicates	candidate goals	consistent goals per update	goals in $\mathcal{S}$ and $\mathcal{G}$ per update	cpu seconds per update	percent work saved
4	64	33	6.5	0.4	67.5
6	256	129	9.98	0.9	67.8
8	1,024	486	13.53	1.3	48.0
10	4,096	1,778	18.89	1.8	67.3
12	16,384	7,913	34	5.1	63.5
14	65,536	26,871	38.44	7.7	71.1
16	262,144	116,596	55.13	12.5	52.2
18	1,048,576	643,919	98.5	15.255	61.2

Table 3: Results from experiments in Unix: BOCE is fast because it enumerates only a small fraction of the plan library. These numbers are the average from over 100 randomized runs. We ran our algorithms, coded in Lisp, on a 134 MHZ SGI Indy.

when these conditions arise, besides conjunctive search goals, is when the actor interleaves goals. Condition (1) holds because the number of combinations quickly explodes. Conditions (2) and (3) hold because (under certain assumptions) $g_i \wedge g_j$ is more general than g_i or g_j. In the degenerate case, however, when (2) or (3) does not hold, BOCE reverts to RIGS and each goal's consistency is individually computed; performance thus becomes linear in the number of goals.

9 Empirical validation

In this section we describe how we tested the BOCE algorithm on realistic data in two domains to show that it works quickly and that its output could be useful.

9.1 Unix

We simulate a world which contains a variety of objects, such as files and printers, and simulate an actor searching for those objects. At each time step the actor executes one action from one of her plans. After each action is observed, the goal recognizer can output necessary or sufficient goals. These goals are used to suggest an object to the actor. For example, if the recognizer outputs that a free, color, second-floor printer would satisfy the actor's goal, and such a printer exists in our simulation, then that printer is suggested to the actor. If the suggested object satisfies one of the actor's goals, then the actor ceases to execute any actions in the plan for that goal.

We generate the action sequence $\mathcal{A}$ by interleaving one or two sequences of actions, each of which represents the execution of a plan to find some object. Some of these sequences were gathered from human subjects solving goals in Unix. In all trial runs, we set $K = 2$, and Λ to be 25 action schemas for various Unix commands. We varied Ω to contain between two and nine predicates for describing printers and two to nine predicates for describing files. The printer goals are decomposable, and the file goals are composable. As we add predicates to Ω, the plan library grows exponentially.

Our prototype implementation in Lisp is reasonably fast. As shown in table 3, BOCE performs recognition on a library containing over a million goals in about 16 cpu-seconds per update. These results suggest we could achieve real-time performance if we rewrote BOCE in optimized C code.

Table 3 shows that BOCE is fast because it enumerates only a fraction of the consistent goals. For 14 predicates, BOCE computes the consistency of 39 goals per update, on average. Maintaining an explicit list of consistent goals would require checking an average of 26,871 goals, per update.

The last column in table 3 quantifies the work saved by task-completion using BOCE, compared to letting the actor solve all her own goals. A 30% savings occurs, for example, if the task-completion agent satisfies the actor's goal after the actor executes 7 actions in a 10 action plan. In our experiments, goal recognition leads to well over 50% work saved.

9.2 Toast

Toast [Agre and Horswill, 1992] is a simulated agent that performs cooking tasks. Toast accepts goals such as making omelettes, cleaning dirty dishes, or setting the table. Toast performs a greedy action-selection that, given certain constraints on the domain, allows it to solve its goals without scheduling any future actions. We extended the default domain to include a variety of goals such as making pancakes with blueberries and omelettes with green peppers and onions.

In each trial run, we gave Toast between 1 and 6 goals from a total of 14 possible goals. Additionally, Toast always pursues a background goal of cleaning the kitchen. As BOCE observes Toast execute actions, BOCE both partially and (with more observations) fully identifies Toast's goals. For example, if Toast pours flour into a bowl, breaks an egg into the bowl, and cleans the cherry pitter then BOCE determines that making pancakes with cherries is necessary for one of Toast's goals, though the pancakes may contain other flavors as well.

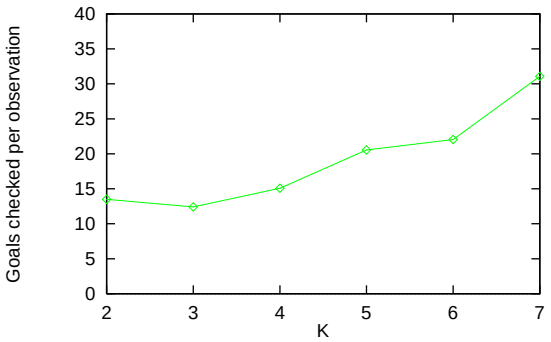

Figure 3: Results from experiments with Toast: The size of the $\mathcal{S}$ and $\mathcal{G}$ boundaries grows linearly with K, the number of goals Toast might pursue simultaneously. Since there were 15 individual goals, the number of goal combinations is 15^K. These numbers are the average over 50 randomized runs.

Figure 9.2 shows that our version space representation remains small even as the number of goal combinations grows exponentially in K. Since there are 15 total goals, there are 15^K combinations of goals that BOCE must consider.

Although we supply the goals, we do not supply the plans. Toast generates the actions to solve its goals. The directed-weak plan bias, described in section **??** and computed by the RIGS algorithm, determines which plans are included in BOCE's plan library $\mathcal{PL}$. If the plans that Toast actually executes are not in $\mathcal{PL}$, then BOCE will produce erroneous conclusions. The fact that our recognizer works in this domain reflects the general applicability of the directed-weak bias.

9.3 Comments

These results may not be encouraging in that the version space grows exponentially with the number of predicates. One can imagine domains with, say, 200 predicates. These domains will be tractable, however, if the predicates can be factored into subgroups. For example, there may be ten predicates for describing printers, ten for machines, and so on. In this case the number of goals would be $(20 \times 2^{10})^K$. For comparison, the largest goal set in our Unix experiments was $(2 \times 2^9)^K$, for $K = 2$.

10 Related work

In order to automate construction of the plan library and achieve fast recognition our approach sacrifices expressiveness of the plan representation. Plan hierarchies, meticulously described by [Kautz, 1987], are a widely-used (e.g. [Ardissono and Cohen, 1995, Lin, 1995, Lin and Goebel, 1990, Vilain, 1990, Wobcke, 1994]) example of a more expressive representation. Plan hierarchies allow one to specify any plan library: arbitrary plans and goals can be omitted and arbitrary constraints posted between steps of plans. In contrast, our approach requires the plan library to contain all and only plans and goals that share an underlying structure, defined by the plan and goal biases.

Others before us have traded expressiveness for tractability. [Vilain, 1990] converts recognition into context free grammar parsing by requiring plan steps to be totally ordered and not shared or interleaved. The complexity of his algorithm is $O(|\mathcal{H}|^2|\mathcal{A}|^3)$ where $|\mathcal{H}|$ is the size, including edges, of the input plan hierarchy and $\mathcal{A}$ is the observations. [Lin and Goebel, 1990] describe a message-passing scheme with complexity $O(|\mathcal{H}||\mathcal{A}|^3)$ under Vilain's restrictions.

Although the faster of these algorithms is linear in $|\mathcal{H}|$, $\mathcal{H}$ contains every candidate goal. For decomposable and composable goal sets, $\mathcal{H}$ will thus contain $2^{|\Omega|}$ goals, where Ω is the set of predicates for forming goals. In contrast, for decomposable goals, BOCE is linear in $|\Omega|$. Furthermore, our experiments provide reason to believe that BOCE will outperform these algorithms for composable goals as well. We tested BOCE on libraries containing millions[4] of goals. In the above systems, this necessarily leads to grammars with millions of rules or message networks with millions of nodes. In contrast, BOCE does not *have to* enumerate every goal. Our experiments show BOCE's boundaries can contain less than 1/1,000 of all consistent goals.

BOCE's consistency algorithm, RIGS [Lesh and Etzioni, 1995], is polynomial in the number of primitive actions, while Lin and Goebel's algorithm is linear in the number of complex actions in $\mathcal{H}$. The number of complex actions can be exponential in the number of primitive actions. More specifically, the complexity of RIGS is $O((|\Lambda| + |\mathcal{A}|)^6|\mathcal{GS}|)$ where $\mathcal{GS}$ is the set of goals to be tested. BOCE's complexity for decomposable goals is thus $O((|\Lambda|+|\mathcal{A}|)^6|\Omega|)$ and for composable goals is $O((|\Lambda| + |\mathcal{A}|)^6 2^{|\Omega|})$. How does this compare to Lin and Goebel's complexity? For our plan and goal biases, $\mathcal{H}$ will necessarily contain $2^{|\Omega|}$ goals and could contain $2^{|\Lambda|}$ complex actions. Thus $\mathcal{H}$ can contain $(2^{|\Omega|} + 2^{|\Lambda|})^2$ edges, making the above algorithm's worst-case running time $O((2^{|\Omega|} + 2^{|\Lambda|})^2|\mathcal{A}|^3)$.

[Weida and Litman, 1992] automate construction of the plan library to the extent that they automatically compute the generality relationships between plans. They also gain considerable efficiency by maintaining only the boundaries within a space of plans. They do, however, require a complete list of all candidate plans as input. In contrast, BOCE constructs its own library

[4] The cross-product of the thousand separate goals in our experiments can be treated as a single goal set of size one million. To allow step sharing and interleaving with the above algorithms, each combination of goals needs to be represented explicitly in $\mathcal{H}$

and exploits generality to avoid fully enumeration of the library and to enable partial recognition. RIGS is similar in spirit to the Unix Consultant [Wilensky *et al.*, 1988], which also chains actions together. However, they do not construct goals (only plans) or define an implicit plan library, as we do, but instead focus on providing advice to Unix users. [Pynadath and Wellman, 1995] work is also concerned with recognition based on beliefs of how the actor plans. Their work focuses on accounting for the context in which plans are generated and uncertainty in recognition, but does not automatically construct or infer the actor's potential plans or goals.

11 Conclusions

New techniques are needed to perform plan and goal recognition in domains in which plan libraries contain tens of thousands of goals. Three of our contributions can be summarized as follows.

First, we automate construction of the plan library. Almost all recognizers require a hand coded plan library. Our goal recognizer constructs its own plan library from a much smaller set of primitives.

Second, we extend techniques from machine learning and planning to provide representations and algorithms for fast recognition. New techniques for fast recognition are of interest because recognition is intended to support real-time interaction with people; new techniques are needed because plan libraries can grow exponentially in the size of the domain. We have identified a class of goals for which BOCE is exponentially faster than other fast recognition algorithms.

Third, our experiments represent a substantial advance in empirical evaluation for plan recognition. We test our goal recognizer on large libraries, use data generated independently of the plan library, and quantify the positive impact of goal recognition.

References

[Agre and Horswill, 1992] P. Agre and I. Horswill. Cultural support for improvisation. In *Proc. 10th Nat. Conf. on AI*, pp. 363–368, 1992.

[Ardissono and Cohen, 1995] L. Ardissono and R. Cohen. On the value of user modeling for improving plan recognition. In *Proceedings of the workshop on The Next Generation of Plan Recognition Systems: Challenges for and Insight from Related Areas of AI*, pp. 8–12, August 1995.

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

[Etzioni, 1991] Oren Etzioni. STATIC: A problem-space compiler for prodigy. In *Proc. 9th Nat. Conf. on AI*, pp. 533–540, 1991.

[Etzioni, 1993] Oren Etzioni. Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62(2):255–302, 1993.

[Fikes and Nilsson, 1971] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4), 1971.

[Huber *et al.*, 1994] J.M. Huber, H.D. Edmund, and M.P. Wellman. The automated mapping of plans for plan recognition. In *Proc. 10th Conf. on Uncertainty in Artifical Intelligence*, pp. 344–51, July 1994.

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

[Lesh and Etzioni, 1995] Neal Lesh and Oren Etzioni. A sound and fast goal recognizer. In *Proc. 15th Int. Joint Conf. on AI*, pp. 1704–1710, 1995.

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

[Lin, 1995] D. Lin. Program recognition by observation. In *Proceedings of the workshop on The Next Generation of Plan Recognition Systems: Challenges for and Insight from Related Areas of AI*, pp. 84–89, August 1995.

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

[Pynadath and Wellman, 1995] D.V. Pynadath and M.P. Wellman. Accounting for context in plan recognition, with application to traffic monitoring. In *Proc. 11th Conf. on Uncertainty in Artifical Intelligence*, pp. 472–81, August 1995.

[Shavlik and Dietterich, 1990] J. Shavlik and T. Dietterich, eds. *Readings in Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1990.

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

[van Beek, 1996] P. van Beek. An investigation of probabilistic interpretations of heuristics in plan recognition. In *Proceedings of the Fifth International Conference on User Modeling*, pp. 113–120, January 1996.

[Vilain, 1990] M. Vilain. Getting serious about parsing plans: a grammatical analysis of plan recognition. In *Proc. 8th Nat. Conf. on AI*, pp. 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.

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

[Wobcke, 1994] W. Wobcke. Plan recognition as belief revision. In *Proceedings of the 1994 Second Australian and New Zealand Conference on Intelligent Information Systems*, pp. 382–386. IEEE, 1994.