Seeing Beyond Shadows: Incremental Abductive Reasoning for Plan Understanding

Ben Meadows (bmea011@aucklanduni.ac.nz)
Pat Langley (patrick.w.langley@gmail.com)
Miranda Emery (meme011@aucklanduni.ac.nz)
Department of Computer Science, University of Auckland
Private Bag 92019, Auckland 1142, New Zealand

Abstract

In this paper we present a new approach to plan understanding that explains observed actions in terms of domain knowledge. The process operates over hierarchical methods and utilizes an incremental form of data-driven abductive inference. We report experiments on problems from the Monroe corpus that demonstrate a basic ability to construct plausible explanations, graceful degradation of performance with reduction of the fraction of actions observed, and results with incremental processing that are comparable to batch interpretation. We also discuss research on related tasks such as plan recognition and abductive construction of explanations.

Introduction

In Plato's famous Parable of the Cave, a person is trapped in a cave where he can perceive only the vague *shapes* of objects that pass in front of a fire, as flickering shadows cast on a wall. The fundamental *forms* of these entities are not directly accessible to the viewer; rather, he must infer them from the available evidence. We wish to borrow two points from this allegory: the perceiver must use vague shadows to infer the nature of the entities casting them, and, given the imperfect and incomplete nature of these inputs, he must sometimes take a 'leap in the dark'.

Plato's parable is more relevant to everyday experience than usually acknowledged, in that we must regularly interpret the goals, beliefs, and intentions of others based on their observed behaviors. We refer to this general task as *plan understanding*. We will discuss related work later, but here we note this problem has received far less attention in the literature than tasks such as *activity recognition* (Aggarwal and Ryoo 2011), which assigns observed behavior to some known category, and *plan recognition* (Goldman, Geib, and Miller 1999), which infers an agent's top-level goals.

In contrast, plan understanding involves the *explanation* of an agent's behavior in terms of its broader mental state. More formally, we can state the task as:

- *Given*: A sequence S of actions that agent A is observed to carry out;
- *Given*: Knowledge about concepts and methods, organized hierarchically, that are available to agent A;
- *Infer*: An explanation, E, organized as a proof lattice, that accounts for sequence S in terms of agent A's goals, beliefs, and intentions.

As its name suggests, plan understanding is a close analogue to language understanding, in that analysis provides a connected account of the input. However, the observation, rather than being a sequence of words, is a sequence of relational actions or events, and the knowledge, rather than being grammatical rules, takes the form of a hierarchical task network (Nau et al. 2001) that comprises a set of conceptual predicates for specifying states and a set of methods that transform them. Moreover, the explanation may not take the form of a proof tree with a single root, but rather a set of interconnected trees that form a lattice.

In the next section, we provide a motivating example that clarifies the task of plan understanding and introduces the challenges it raises for both humans and computational systems. After this, we describe a novel approach to plan understanding, along with an abductive reasoning system that implements its basic tenets: Understanding Mechanism for aBductive Reasoning Agents, or UMBRA. Next we present some claims about UMBRA's capabilities and some experiments that support them. In closing, we discuss related work on abduction, plan understanding, and connected topics, along with directions for future research. Although others have reported frameworks for plan understanding, we believe that our approach incorporates novel features that contribute to our understanding of this important problem.

A Motivating Example

We can clarify the task of plan understanding through an illustrative example. Consider a sequence of observations from the Monroe County corpus (Blaylock and Allen 2005):

> Truck driver $tdriver1$ navigates the dump truck $dtruck1$ to the location $brightondump$, where a hazard team $ht2$ climbs into the vehicle. $tdriver1$ then navigates $dtruck1$ to the gas station $texaco1$, where $ht2$ loads a generator $gen2$ into $dtruck1$.

Based on these observations and on knowledge about the types of goals and activities that arise in such settings, we want to not only infer the top-level activities being pursued, but also to explain how these activities relate to sub-activities and how the latter depend on the situations that occur along

the way. We will assume that an explanation comprises an interconnected set of conceptual and method instances that are organized in a lattice. Any input observation that participates in this lattice is thereby explained; an observation that is connected to more parts of the lattice is more integral to the explanation.

Table 1 presents one explanation for the observations in the example given above. We mark the input actions with italics (the $climb\text{-}out$ action, marked with an asterisk, is a case of partial input requiring an additional assumption). Thus, a substantial portion of the plan's details must be inferred. Elements such as $person(tdriver1)$ must be inferred from taxonomic axioms stating that objects that are drivers are adults and objects that are adults are people. Even given such axioms, there are necessarily gaps in this explanation. For instance, the input does not specify the starting location of $dtruck1$ before it is driven to $brightondump$.

This example raises a number of issues that deserve attention. First, the inputs includes information about both actions and states. For instance, $dtruck1$ may be observed at $brightondump$ in addition to the $navigate\text{-}vehicle$ action that got it there. Second, as we mentioned, some actions may not be observed; in this case, the sequence omits the action ($ht2$ climbs out of $dtruck1$ at $texaco1$). Finally, there may not be enough information to infer a single, top-level goal or activity. In this example, one might conclude that the task $get\text{-}in(gen2, dtruck1)$ has been carried out, but not where the generator is being moved to or for what purpose. Together, these factors impose constraints on the task of plan understanding. We will return to such issues as we present our approach to this problem, to which we now turn.

Table 1: Sample explanation for a *get-in(gen2, dtruck1)* subplan. Italicized elements are operator-level actions, with asterisks marking actions that are elided in the input sequence.

get-in(gen2, dtruck1)
- ← get-to(ht2, texaco1)
  - ← get-to(dtruck1, br-dump)
    - ← drive-to(tdriver1, dtruck1, br-dump)
      - ← at-loc(dtruck1, _)
      - ← at-loc(tdriver1, _)
      - ← *navigate-vehicle(tdriver1, dtruck1, br-dump)*
        - ← person(tdriver1)
        - ← vehicle(dtruck1)
        - ← can-drive(tdriver1, dtruck1)
        - ← at-loc(dtruck1, br-dump)
        - ← at-loc(tdriver1, br-dump)
  - ← get-in(ht2, dtruck1)
    - ← not(non-ambulatory(ht2))
    - ← person(ht2)
    - ← *climb-in(ht2, dtruck1)*
      - ← at-loc(ht2, br-dump)
      - ← at-loc(dtruck1, br-dump)
      - ← fit-in(ht2, dtruck1)
      - ← at-loc(ht2, dtruck1)
  - ← get-to(dtruck1, texaco1)
    - ← drive-to(tdriver1, dtruck1, texaco1)
      - ← at-loc(dtruck1, br-dump)
      - ← at-loc(tdriver1, br-dump)
      - ← *navigate-vehicle(tdriver1, dtruck1, texaco1)*
        - ← person(tdriver1)
        - ← vehicle(dtruck1)
        - ← can-drive(tdriver1, dtruck1)
        - ← at-loc(dtruck1, texaco1)
        - ← at-loc(tdriver1, texaco1)
  - ← get-out(ht2, dtruck1)
    - ← not(non-ambulatory(ht2))
    - ← person(ht2)
    - ← *climb-out(ht2, dtruck1)* *
      - ← at-loc(ht2, dtruck1)
      - ← at-loc(dtruck1, texaco1)
      - ← at-loc(ht2, texaco1)
- ← can-lift(ht2, gen2)
- ← *load(ht2, gen2, dtruck1)*
  - ← at-loc(gen2, texaco1)
  - ← at-loc(dtruck1, texaco1)
  - ← at-loc(ht2, texaco1)
  - ← fit-in(gen2, dtruck1)
  - ← at-loc(gen2, dtruck1)

Plan Understanding through Abduction

Our scientific aim is a computational account of plan understanding on tasks like the example just presented. In this section, we describe UMBRA, an implemented system that addresses this class of tasks. After stating our core theoretical assumptions, we consider the content of the system's working memory and knowledge base. After this, we examine the mechanisms that operate over them, including the manner in which they guide search through the space of alternative explanations.

Theoretical Tenets

In general, we may distinguish a theoretical framework from its particular implementations. We can characterize our theory of plan *understanding* in terms of four key assumptions that differentiate it from approaches to plan *recognition*:

- *Plan understanding involves inference about the participating agents' mental states*. These states include not only agents' beliefs and goals about activities and actions, but also about the environmental situations.
- *Plan understanding involves the generation of plausible explanations, and thus is inherently abductive in character*. That is, it posits default assumptions about agents' beliefs and goals that, together, account for observations.
- Because observations about agents' activities arrive sequentially, *abduction operates in an incremental fashion*. Thus, only a few observations are interpreted at a time, with later inferences building on ones introduced earlier.[1]
- Because plan understanding arises from observations about agents' activities, *the abduction process operates in a data-driven manner*. That is, inference mainly involves 'bottom-up' chaining off observations, rather than being driven by 'top-down' chaining from queries.

[1]The experimental studies in a later section use batch processing to provide a baseline, but we will see that our system supports incremental processing.

Combined with the task-related challenges discussed earlier, these four assumptions place constraints on our computational account of plan understanding.

In the remainder of this section, we describe UMBRA, an implemented system that incorporates these theoretical tenets. We have implemented the system and knowledge base in SWI-Prolog (Wielemaker et al. 2012) because it enables reasoning over relational and hierarchical structures, and because its ability to track dynamic global variables supports a working memory that changes over time.

Representational Assumptions

UMBRA incorporates a knowledge base that it encodes as a set of rules stated in predicate logic, and a working memory that contains a set of ground literals. Knowledge takes the form of a hierarchical task network that describes both primitive operators and decomposition rules in a STRIPS-like (Fikes and Nilsson 1971) notation, with each of the latter specifying a task and a body that contains conditions, invariants, subtasks, and effects. For instance, UMBRA's knowledge may include multiple methods for decomposing the task $remove\text{-}wire$ into subtasks in different situations. The system encodes these hierarchical structures as Prolog clauses. Primitive tasks correspond to operators that achieve their effects without further decomposition. The knowledge base also contains inference rules that specify high-level state predicates in terms of lower-level ones, including taxonomic rules, such as $vehicle(X) \leftarrow ambulance(X)$.

Working memory in UMBRA consists of beliefs held by the primary agent that is attempting to understand others' behaviors. For example, the working memory element $belief(agent1, hospital(park\text{-}ridge))$ denotes that agent1 believes $park\text{-}ridge$ is a hospital. The agent may also hold negated beliefs. In addition, each working memory element has a start time that indicates when the agent adopted it and an end time that indicates when it was discarded.

UMBRA takes as input a sequence of observations encoding the primary agent's beliefs that particular activities or state literals have occurred, along with their start and end times. For example, the agent may observe that another person has received emergency treatment at a hospital or that a site does not have electricity. These typically involve low-level actions or relations, although the system can be told that higher-level tasks or relations have taken place. Initial working memory also includes common domain facts, such as particular objects being people, snow plows, or food.

The system's output is an explanation of the observations. This account takes the form of interconnected inferences in working memory, many of which constitute default assumptions, along with information about the rule instances that produced them. The inferred literals correspond to instantiated versions of rule heads, conditions, effects, invariants, and subtasks. Some literals include Skolem terms reflecting variables that were unbound when the belief was generated.

Generating Extensions to Explanations

UMBRA utilizes its knowledge in an iterative, data-driven manner that constructs an explanation of observations incrementally as they enter working memory. This process is abductive in character in that, when applying an inference rule, it can introduce default assumptions that are not deductively valid but are nevertheless plausible.

The system operates in a series of *observational cycles*, on each of which it receives zero or more observations (with associated time stamps) that are followed by zero or more *inference cycles*. This process continues until no inputs arrive, at which point it halts. During each observational cycle, UMBRA applies inference rules sequentially until it exceeds a cost threshold T_{cost} (described below). By the end of each observational cycle, the system has selected zero or more rules to apply, partially matched them against elements in working memory, and added default assumptions – based on unmatched literals in the rules' heads and antecedents – to the same memory.

On each inference cycle, UMBRA attempts to instantiate and apply rules to extend its current explanation. To this end, it identifies every rule R with antecedents that unify with some element E that is either an input or an inferred rule head. For each such pair, UMBRA generates a partially instantiated head for R based on its antecedents' bindings to E. Each instantiated head serves as a seed for a candidate rule application.

The system expands each seed by finding which of the rule's antecedents unify with elements in working memory, considering all maximal partial matches if more than one exists. For every partial match, UMBRA generates tentative default assumptions to fill in the remainder, rejecting any candidates that would introduce more assumptions than a user-specified maximum. We will refer to each candidate that meets this criterion, including the instantiated head, the set of matched antecedents, and the set of unmatched assumptions, as a *rule extension*. Each extension corresponds to one way that UMBRA might elaborate its current explanation, but it must still select which of them to utilize on the current inference cycle.[2]

Evaluating Candidate Extensions

UMBRA employs a measure of *cost* C to select among its candidate rule extensions, with lower scores being more desirable. This evaluation function has three components which penalize:

- candidates with antecedents and heads that unify with fewer elements already present in working memory (E);
- alternatives that would introduce more new default assumptions (D); and
- rule extensions that unify with working memory elements that were, on average, less recently added (R).

UMBRA combines these component measures in an arithmetic function, with the weight on each factor specified by the user. In the runs reported in this paper, we used the function $C = 10 \cdot E + D + R/200$. This gives the greatest weight to incorporation of working memory elements into the ex-

[2]We can contrast this approach with that taken by AbRA (Bridewell and Langley 2011), which explicitly picks a particular literal or 'focal belief' from which to chain before it selects a rule instance through which to chain.

planation and the next highest to the number of default assumptions, with recency used only to break ties.

We should note that the first two components bias the system toward more *coherent* explanations, in the sense that Ng and Mooney (1992) propose. These also reflect Thagard's (1978) criteria of *consilience*, which favors explaining as much as possible, and *simplicity*, which favors as few assumptions as possible. The third component gives the system a recency bias similar to that in some production-system architectures (e.g., Forgy 1981).

Applying Selected Rule Extensions

Once UMBRA has generated and evaluated a set of rule extensions, the final step in each inference cycle applies the selected candidate. However, before this occurs, the system compares the cumulative cost if the rule instance were applied to the system's cost threshold T_{cost}. This threshold places a limit on the total number of assumptions UMBRA will make on a single observable cycle. If the number of default assumptions associated with a rule instance would cause the total to exceed T_{cost}, the system does not apply the rule and it ends the observational cycle.

Otherwise, UMBRA applies the rule instance, which involves adding both the associated head and default assumptions to working memory. On the next inference cycle, these new elements are available to unify with candidate rules. In this way, the explanation process is driven not only by external content, but also by new inferences. This process can also lead the system to replace variables in existing working memory elements with constants that appear in other literals, as new information becomes available about variable bindings. Thus, over time, UMBRA elaborates its explanation to cover an increasing number of external inputs.

At any given point, there is no guarantee that UMBRA has inferred a top-level task or goal. By default, the system runs until it receives no further inputs. Each observational cycle continues until T_{cost} is reached, at which point it is reset for future inputs. However, UMBRA also incorporates a *pause condition* that halts abductive reasoning temporarily when all inputs are covered *and* when it has inferred at least one top-level task. This immediately ends the current observational cycle, so the system performs no more processing until it receives more inputs. The purpose is to prevent the abduction engine, upon reaching a sufficiently full explanation, from needlessly extending it with further assumptions.

Empirical Evaluation

Although we maintain that UMBRA reflects a novel and promising approach to plan understanding, whether it performs this task effectively is an empirical question. In this section, we report experiments that evaluate three distinct claims about the system:

- Given observed actions and relevant knowledge, UMBRA generates reasonable inferences that explain these actions in terms of high-level tasks;
- UMBRA's ability to generate reasonable explanations degrades gracefully as one reduces the percentage of observed actions given as input, so that it can still make reasonable inferences from partial information; and
- Incremental processing of observations that arrive sequentially leads to explanations that are comparable to ones produced by batch processing when all observations are available at the outset.

To make these hypotheses operational, we must specify performance measures that let us evaluate the system's performance. Because we are interested in the entire explanation that UMBRA produces, not just the top-level action, we have adopted the common measures of *precision* and *recall*.

Given a generated explanation with N elements and a target explanation with M elements, where elements are those literals in the structure other than inputs, we can calculate the number of true positives T, the number of false negatives or errors of omission as $O = M - T$, and the number of false positives or errors of commission as $C = N - T$. We can then compute the precision as $T/(T + C)$ and the recall as $T/(T + O)$. We believe that finer-grained measures of this sort are more appropriate for plan understanding than the accuracy metric that is typically reported in the literature on plan recognition, which focuses exclusively on the top-level goal.

Following Raghavan and Mooney (2010), we calculated true positives by awarding one point for inference of a correct predicate and additional points for each correct argument within a correct predicate. We computed false negatives by awarding one point for each variable in a correct predicate and, for entirely missing predicates, one point plus the arity of the predicate. Following Bridewell and Langley (2011), we extended this scoring system to all tasks in the plan structure, not just the top-level predicate. We did not include state literals in the scores, only tasks and operators.

The Monroe County Domain

For our preliminary tests of these hypotheses, we used the Monroe domain (Blaylock and Allen 2005), which was designed for plan recognition and which is familiar to many researchers in the area. Typical elements include blocked roads, environmental hazards, injuries, humans (police units, victims, different types of work crew), and various purpose-specific vehicles. The domain includes many types of locations, from towns to hospitals and from roads to gas stations, along with abstract entities such as power or water suppliers.

Hierarchical knowledge for this domain consists of 51 methods for decomposing 38 distinct tasks, and 41 primitive operators. There are ten top-level tasks: setting up a shelter; fixing a water main; clearing a hazard such as a chemical spill from the road; clearing a wrecked vehicle from the road; clearing a fallen tree from the road; plowing snow from a road; quelling a riot; providing temporary heat to a person; fixing a power line; and providing an injured person with medical attention. The methods specify ways (possibly more than one) to decompose these tasks into subtasks, in a hierarchical manner that terminates in the operators. The knowledge base also includes 55 taxonomical rules, such as "an object X that is a member of a work crew is an adult".

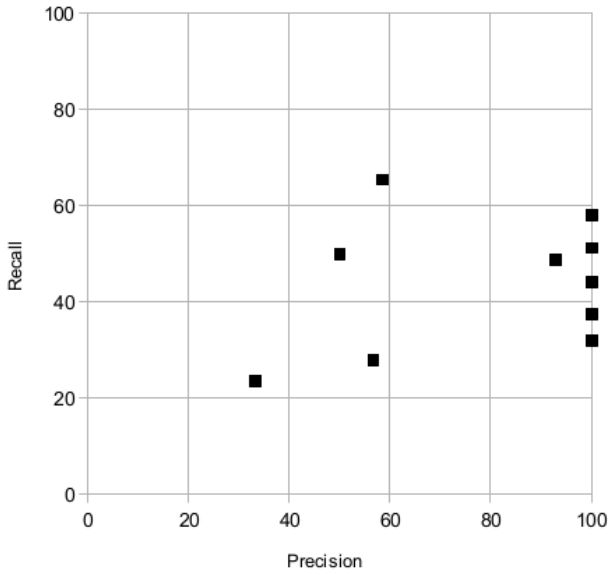

Figure 1: Comparison of the precision and recall scores for each problem over ten 'batch' experimental runs.

Experimental Design

For our initial experiments, we selected ten problems from the Monroe domain, one for each of the ten top-level plan predicates in the corpus. We ran UMBRA with a cost threshold T_{cost} of 30, a default assumption cap of 20 per inference cycle, and the simple evaluation metric that we described in the previous section.

Our first study involved running the system in batch mode, with all of the actions available, on each example problem, and recording the precision and recall for each task. In our second experiment, we varied this procedure by removing increasing portions of the input sequence, following Raghavan and Mooney's (2010) scheme. More specifically, we provided UMBRA with the first 100%, 75%, 50%, and 25% actions as input. In the third study, we compared the system's behavior when provided with all actions (batch) with the same observations presented sequentially and processed incrementally. Here we allocated the same amount of processing effort (in terms of number of inference cycles) for both conditions.

Experimental Results

Our initial experiment with UMBRA produced reasonable results on the basic plan understanding task. Each test covered between five and 22 observational cycles, depending on the number of inputs given. The explanations generated as output had a mean precision of 79.8% and mean recall of 44.1% based the true plan skeletons. UMBRA behaved somewhat differently on different top-level plans; in the worst case, its scores were 33.3% for precision and 23.5% for recall. Because of the way that UMBRA builds explanations, all rule applications inferred the accompanying conditions for those rules, with Skolem variables as needed.

Table 2: Precision and recall for the non-incremental and incremental cases. Measurements are presented as percentages, with standard errors following them.

Input	**Precision**	**Recall**
Batch 100%	79.8 ± *8.3*	44.1 ± *4.3*
Batch 75%	79.4 ± *9.8*	41.4 ± *4.9*
Batch 50%	70.4 ± *11.5*	29.0 ± *4.7*
Batch 25%	55.2 ± *12.6*	16.4 ± *2.3*
Incremental	70.6 ± *10.2*	53.6 ± *4.3*

Figure 1 plots precision against recall for each test problem. Note that, for half of them, UMBRA achieved 100% precision, in that it generated no false positives. Combined with the low recall rate, this suggests this system errs in being overly conservative. Note also that there is some correlation between our two performance measures; some plans were simply harder to understand than others. Analysis suggests that this may result from details of the Monroe knowledge base, in which some domain predicates lack the arguments needed for complete plan understanding.

Our second claim was that UMBRA's performance would degrade gracefully as one reduced the percentage of observations available. The precision and recall scores in Table 2, for varying amounts of elided input, generally support this hypothesis. Note that UMBRA performs nearly as well, on both dependent measures, with 75 percent of the actions as with full sequences, suggesting that its abductive inference mechanisms are working as intended. Even when only 25 percent of the actions are known, precision remains at the 55 percent level, although recall drops to only 16 percent.

The final claim was that UMBRA's performance with incremental processing is comparable to that in batch mode. The results at the bottom of Table 2 are roughly consistent with this claim, with the precision for incremental abduction being somewhat lower and recall somewhat higher. However, standard errors are high, presumably due to differences in problem difficulty. The plots in Figure 2 provide more insight and show that both scores are generally comparable but that, indeed, the incremental version has slightly lower precision and slightly higher recall.

One plausible account for this behaviour is related to UMBRA's bias toward fewer default assumptions, and thus toward applying inference rules with fewer unsupported antecedents. Such rules have a lower chance of selection throughout a run in batch mode, but they have a higher chance during early stages of incremental processing, when very few rules may match at all against the current input. Thus, the incremental version may have more effective recall when plans include rules with many conditions.

Additional Analyses

In order to better understand UMBRA's behavior, we carried out additional analyses. For instance, we were interested in the number of rules the system applied correctly but

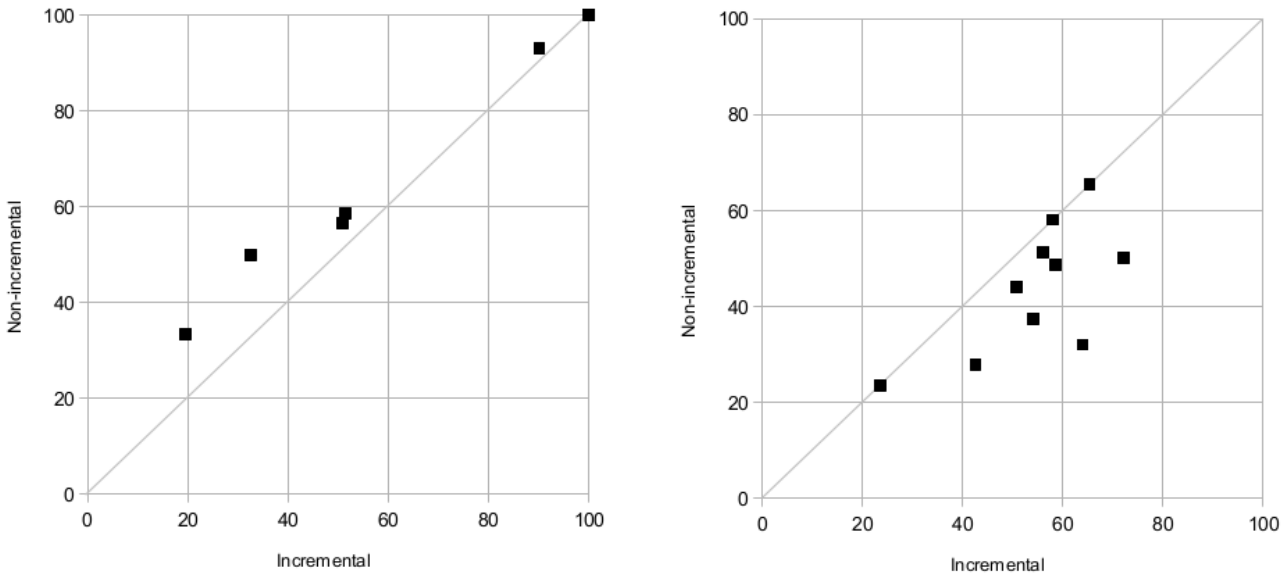

Figure 2: Comparison of precision (left) and recall (right) for ten incremental and nonincremental runs, with the line $x = y$ shown for comparison. Incremental processing in UMBRA typically leads to lower recall but higher precision.

with incorrect assumptions. This occurred fairly often in the Monroe domain, where rule heads are sparse and unspecified world states lead to many free variables in rule bodies. For this purpose, we defined false positives as rule instances with at least one state or action condition that does not appear in the target plan. We measure these as a percentage of the number of rules successfully applied in a run.

We are interested in these for two reasons. First, applied rules may make poor assumptions in their bodies yet still have correct rule heads. Second, these spurious rules provide a metric for *understanding*, as opposed to overall and head-level classification. Table 3 shows results with this revised metric. Note that the number of rules inferred decreases as less input becomes available to chain off. The number of spurious rule applications appears to be relatively stable: removing inputs does not make the system any more likely to make guesses.

In addition, we varied the parameters that regulate the number of times UMBRA may make assumptions: the cost threshold T_{cost} and the default assumption cap. Decreasing these values tends to reduce the number of rules applied, and thus reduces the number of false negatives. Increasing them drives the system toward inferring an entire plan, but only at the cost of increased run time and diminished solution quality. In batch mode with unbounded resources, the system often constructs a complete explanation the top level on a single cycle, populated more or less entirely by assumptions and Skolem variables.

We also discovered that UMBRA found certain things more difficult to make good inferences about. For example, the three plans on which the system performed worst featured the 'call' action, which denoted a phone call to some entity. An agent can use this to shut water or electricity on or off, or to declare a curfew, at many different locations that the $call$ predicate does not actually specify. Combined with the fact that most of the rules involving $call$ have few conditions and thus relatively few extra assumptions, this resulted in a substantially increased rate of false positives.

However, UMBRA also did well on some additional tasks. Informal studies with interleaved plans suggested that explanation quality was similar to situations in which the system observed the two plans separately, although the run time was considerably longer. This merits further investigation, as one of UMBRA's key features is the ability to construct explanations that lack a single root node.

In summary, our experiments suggest that our system can generate coherent explanations of observed action sequences from hierarchical knowledge, that it forms reasonable explanations even when some observations are withheld, and that incremental processing produces explanations of comparable quality to those generated from batch analysis.

Related Research

Our approach to plan understanding builds on a large body of related research in the related areas of plan recognition and abductive inference, but it also incorporates some distinctive features that contribute to this extensive literature. Space prevents us from covering all work in these two areas, so here we review the most closely related efforts.[3]

One recurring theme that UMBRA incorporates is the hierarchical representation of activities. For example, Blaylock and Allen's (2005) Monroe corpus assumes that plan knowledge is encoded as a hierarchical task network, and their approach to plan recognition operates over these struc-

[3]We will not discuss work on activity recognition (Aggarwal and Ryoo 2011), as it typically adopts non-relational representations, and it seldom incorporates hierarchical activities.

Table 3: Rates of false assumption used as support. The table reports, for the batch and incremental cases, the mean number of rules the system successfully inferred, and the percentages of these rules whose conditions contained at least one false positive, along with the standard errors.

	Mean number of rules inferred	**Rule bodies with false assumptions**
Batch 100%	14.8	58.1% ± *7.5*
Batch 75%	11.9	42.0% ± *9.7*
Batch 50%	7.9	46.8% ± *12.2*
Batch 25%	5.3	54.7% ± *12.9*
Incremental	17.8	52.8% ± *6.9*

tures. Other researchers (Kautz and Allen 1986; Goldman et al. 1999; Raghavan and Mooney 2010) have adopted similar notations, so that inferences about agent intent often take the form of a proof tree.

As we have noted, most work has focused on inferring the top-level goal or activity, whereas UMBRA instead posits an interconnected explanation that includes intermediate states and activities. However, this emphasis may have resulted partly because it is easier to measure the accuracy in terms of top-level goals than evaluating complete explanations. We believe some existing methods for plan recognition could be extended to support plan understanding as we define it.

Many approaches to plan recognition, at least ones that operate over hierarchical activities, adopt query-driven or top-down methods. This is a natural response given the clear analogy between hierarchical task networks and logic programs, which are often associated with query-driven interpreters. In contrast, UMBRA incorporates a data-driven approach that chains off observations and constructs an explanation from the 'bottom up'. It shares this behavior with Bridewell and Langley's (2011) AbRA, which served as an inspiration for our work. One advantage of this method is that it directly supports explanations that do not involve a single top-level activity, either because there is not enough information to infer it (e.g., at an early stage in incremental processing) or because the observations reflect a number of distinct interleaved plans.

Another common technique is to propositionalize or otherwise transform the domain at compile time – for instance, by enumerating all possible ground literals (Kautz and Allen 1986; Appelt and Pollack 1992; Ramirez and Geffner 2009, 2010). The advantage is that this lets one utilize established methods for probabilistic inference, which easily handle missing information, over the resulting structures. The disadvantage is that the number of possible ground literals grows exponentially with the number of predicates and objects, raising problems with scaling to large domains. In contrast, UMBRA generates candidate ground literals in a local, limited way at inference time. This should scale much better than enumerative methods, and our experimental studies suggest it remains effective at generating explanations.

We have been strongly influenced by previous research on abductive reasoning. Hobbs et al. (1988) focused on sentence interpretation rather than plan understanding, but their approach introduced plausible assumptions to produce explanations of input. Ng and Mooney (1992) introduced the use of coherence to guide search for candidate explanations, an idea that we have incorporated into UMBRA. And we have already noted our indebtedness to Bridewell and Langley's AbRA, which also adopted an incremental, data-driven approach to abduction that constructs coherent explanations. UMBRA differs from its predecessor mainly by omitting a separate step for selecting a literal to chain off, by utilizing Prolog's mechanisms for unifying variables and Skolems during the inference process, and by using more limited lookahead during its search for an explanation.

Other work somewhat close to our own is that of Goldman, Geib, and Miller (1999), whose framework can handle partially-ordered plans and plan interleaving, information from 'context' (i.e., state-related literals), and actions by multiple agents. Also, Appelt and Pollack (1992) used abductive inference to generate complete explanations of agents' observed behavior; their system associates weights with each rule's antecedent while ours uses an evaluation measure for particular rule extensions, but both attempt to find low-cost proofs that explain observations with default assumptions as needed. We should additionally mention Waern's (1994) theoretical work on incremental approaches to plan recognition, which shares with UMBRA the treatment of early default assumptions as givens for later processing. Finally, the early work of Kautz and Allen (1986) in plan recognition emphasizes the usefulness of a system that can generate partial plans, and supports cases where an action is performed as part of multiple concurrent plans.

Plans for Future Work

Although our experiments with UMBRA produced encouraging results, there are several areas in which we should extend it. One obvious direction involves demonstrating the system's generality on a number of different domains. We should also test the program's behavior when we provide it with sequences of environmental states rather than actions, as this more closely models the setting in which humans carry out plan understanding.

Another limitation is that, although UMBRA already associates start and end times with action and state literals, it currently uses them only in checking the immediate applicability of rules. In future work, we should extend the system to support temporal reasoning that can be revised in the case of acquisition of new information about temporal or ordering constraints. In addition, we should incorporate support for belief revision, so that UMBRA can handle cases in which it makes incorrect default assumptions that it must later retract when it leads to contradictions. This should improve the system's ability to carry out incremental construction of explanations.

Furthermore, we should extend our system to incorporate weights on domain predicates to support different costs for default assumptions, as already done by Appelt

and Pollack's (1992) weighted abduction and by Goldman et al.'s (1999) probabilistic abduction. We should also explore adding weights to inference rules themselves, as Waern (1994) has proposed, to reflect the costs or probabilities of alternative decompositions.

Finally, we hope to extend our framework to explain interactions among agents. We are especially interested in plan understanding that involves social cognition. This will require not only making inferences about participating agents beliefs about the environment, but about their beliefs about others' beliefs. An augmented version of UMBRA should represent and understand plans that involve social interactions with deception, persuasion, and even instruction.

Concluding Remarks

In this paper, we reviewed the task of plan understanding, which involves generating an explanation of observed agents' actions, not only in terms of their top-level activities but in terms of intermediate activities and their associated states. We noted a number of challenges that this task presents and outlined a theoretical framework for addressing it. We then described UMBRA, an implementation of this framework that utilizes incremental, data-driven abduction to plan understanding.

In addition, we reported experimental studies of the system's operation on action sequences from the Monroe corpus. These suggested that UMBRA makes reasonable inferences when the entire sequence is provided, that its performance degrades gracefully as one reduces the percentage of observed actions, and that incremental processing fares nearly as well as batch processing. We compared UMBRA to earlier work on related tasks such as plan recognition and abductive inference, and we proposed some avenues for additional research. Our analyses suggest that, although our approach has much in common with previous efforts, it also incorporates some distinctive features that hold potential for robust and scalable plan understanding.

Acknowledgements

This research was supported in part by Grant N00014-10-1-0487 from the Office of Naval Research. We thank Paul Bello, Will Bridewell, Nick Cassimatis, and Alfredo Gabaldon for discussions that influenced the approach to plan understanding we have reported here.

References

Aggarwal, J. K., and Ryoo, M. S. 2011. Human Activity Analysis: A Review. *ACM Computing Surveys* 43(3): 16:1–16:43.

Appelt, D. E., and Pollack, M. E. 1992. Weighted Abduction for Plan Ascription. *User Modeling and User-Adapted Interaction* 2: 1–25.

Blaylock, N., and Allen, J. 2005. Generating Artificial Corpora for Plan Recognition. In L. Ardissono, P. Brna, and A. Mitrovic, eds., *User Modeling 2005, Lecture Notes in Artificial Intelligence vol. 3538*, 179–188. Edinburgh: Springer.

Bridewell, W., and Langley, P. 2011. A Computational Account of Everyday Abductive Inference. In Proceedings of the Thirty-Third Annual Meeting of the Cognitive Science Society, 2289–2294. Boston, Mass.: The Cognitive Science Society.

Clancey, W. J. 1984. Classification Problem Solving. In *Proceedings of the Fourth National Conference on Artificial Intelligence,* 49–54. Menlo Park, Calif.: AAAI Press.

Fikes, R., and Nilsson, N. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2: 189–208.

Forgy, C. L. 1981. OPS5 User's manual, Technical Report, CMU-CS-81-135, Carnegie Mellon Univ.

Goldman, R. P., Geib, C. W., and Miller, C. A. 1999. A New Model of Plan Recognition. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence,* 245–254. San Francisco, Calif.: Morgan Kaufmann.

Hobbs, J. R., Stickel, M., Martin, P., and Edwards, D. 1988. Interpretation as Abduction. In Proceedings of the Twenty-Sixth Annual Meeting of the Association for Computational Linguistics, 95–103. Buffalo, New York: Association for Computational Linguistics.

Kautz, H., and Allen, J. F. 1986. Generalized Plan Recognition. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, 32–38. Menlo Park, Calif.: AAAI Press.

Leake, D. 1993. Focusing Construction and Selection of Abductive Hypotheses. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence,* 24–29. San Francisco, Calif.: Morgan Kaufmann.

Nau, D. S., Cao, Y., Lotem, A., and Muñoz-Avila, A. 2001. The SHOP planning system. *AI Magazine* 22: 91-94.

Ng, H. T., and Mooney, R. J. 1992. Abductive Plan Recognition and Diagnosis: A Comprehensive Empirical Evaluation. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning,* 499–508. Morgan Kaufmann.

Raghavan, S., and Mooney, R. 2010. Bayesian Abductive Logic Programs. In *Proceedings of the AAAI-10 Workshop on Statistical Relational AI,* 82–87. Atlanta: AAAI Press.

Ramirez, M., and Geffner, H. 2009. Plan Recognition as Planning. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence,* 1778–1783. San Francisco: Morgan Kaufmann.

Ramirez, M., and Geffner, H. 2010. Probabilistic Plan Recognition Using Off-the-Shelf Classical Planners. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. Menlo Park, Calif.: AAAI Press.

Thagard, P. R. 1978. The Best Explanation: Criteria for Theory Choice. *The Journal of Philosophy* 75(2): 76–92.

Waern, A. 1994. Plan Inference for a Purpose. In *Proceedings of the Fourth International Conference on User Modeling*, Hyannis, Mass.: Mitre Corp.

Wielemaker, J., Schrijvers, T., Triska, M., and Lager, T. 2012. SWI-Prolog. *Theory and Practice of Logic Programming* 12(1–2): 67–96.