Abduction in Annotated Probabilistic Temporal Logic*

Cristian Molinaro, Amy Sliva, and V. S. Subrahmanian

Department of Computer Science and UMIACS, University of Maryland
College Park, MD 20742, USA
{molinaro,asliva,vs}@umiacs.umd.edu

Abstract

Annotated Probabilistic Temporal (APT) logic programs are a form of logic programs that allow users to state (or systems to automatically learn) rules of the form "formula G becomes true K time units after formula F became true with L to U% probability." In this paper, we develop a theory of abduction for APT logic programs. Specifically, given an APT logic program Π, a set of formulas H that can be "added" to Π, and a goal G, is there a subset S of H such that $\Pi \cup S$ is consistent and entails the goal G? In this paper, we study the complexity of the Basic APT Abduction Problem (BAAP). We then leverage a geometric characterization of BAAP to suggest a set of pruning strategies when solving BAAP and use these intuitions to develop a sound and complete algorithm.

1998 ACM Subject Classification I.2.3 Logic Programming, Probabilistic Reasoning

Keywords and phrases Probabilistic Reasoning, Imprecise Probabilities, Temporal Reasoning, Abductive Reasoning

Digital Object Identifier 10.4230/LIPIcs.ICLP.2011.240

1 Introduction

Statements of the form "formula G becomes true K time units after formula F became true with L to U% probability" occur naturally in many different domains. For example, it is common to talk about the probability that certain stock prices will increase or decrease within a given amount of time after an economic announcement is made by the US Federal Reserve. In the same vein, it is normal to say that there is a probability that certain diseases will occur at specific levels within some amount of time after a natural disaster has occurred. And in an all too common scenario for air-travelers, there is a certain probability that a plane will take off within a time T after pushing back from the gate. All of these examples can naturally be expressed as Annotated Probabilistic Temporal (APT) rules [19].

In this paper, we consider the problem of *abduction* in APT logic programs (APT-LPs). There are many cases where abduction in such logic programs is critically needed. For instance, [19] describes how APT logic programs describing the temporal and probabilistic behavior of over 30 terrorist groups were automatically extracted from data about those groups. In those APT logic programs, there are two types of predicate symbols—*action* predicates describing actions the group takes (e.g., suicide bombing, kidnappings), and *environmental* predicates describing the environment in which the group operates (e.g., whether the group gets financial support from a foreign state, whether the group can participate in the electoral process in its country, etc.). In such an application, APT-rules have action atoms in the rule

* Some of the authors were partly supported by ARO grants W911NF0910206 and W911NF0910525.

Technical Communications of the 27th International Conference on Logic Programming (ICLP'11).
Editors: John P. Gallagher, Michael Gelfond; pp. 240–250

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1) $interOrgConflict(Group_1) \overset{efr}{\rightsquigarrow} armedAttack(Group_1) : [2, 0.85, 0.95]$
$Group_1$ will use armed attacks as a strategy within two time units of being involved in inter-organizational conflict with a probability between 0.85 and 0.95.

2) $militaryWing(Group_1) \overset{pfr}{\hookrightarrow} bomb(Group_1) : [2, 0.86, 0.91, 0.7, 0.8]$
$Group_1$ will carry out a bombing exactly two time units after establishment of a standing military wing with a probability between 0.86 and 0.91.

Figure 1 A sample APT program based on automatically extracted rules from [19] modeling the behavior of a terrorist organization $Group_1$.

head and environmental atoms in the rule body. Figure 1 provides a small set of such rules. Clearly, defense experts are interested in understanding how they can modify the true set of environmental atoms (subject to constraints specifying what can and cannot be made true) so that a given goal (e.g., reduction of suicide attacks by 10% or more in the next quarter) can be achieved, if possible. In a second application we are building, we are studying the relationship between numerous social-cultural-economic and policy variables with educational outcomes in over 220 countries.[1] Here again, APT rules can describe the temporal relationship between environmental atoms (e.g., percentage of education budget spent on primary education, student-teacher ratio, provision of childcare facilities) and outcomes (e.g., percentage of females enrolled in primary education), together with the probability that those relationships hold. Here, policy makers want to understand what environmental atoms they can make true (subject to some constraints) so that certain desirable outcomes occur during a given time frame with some probability.

In this paper, we consider triples of the form $\langle \Pi, H, g \rangle$ where Π is an APT logic program, H is a set of formulas that can be added (e.g., adding some childcare facilities might be possible, but increasing the share of the education budget spent on primary education might not), and g is a goal we wish to achieve (e.g., improve primary education female enrollment by 10%). The *Basic APT Abduction Problem* (BAAP) tries to find (if possible) a set $S \subseteq H$ such that $\Pi \cup S$ is consistent and entails the goal. In the case of the terrorism and education applications mentioned above, such sets S correspond to things we could do to try to ensure the desired goal occurs at the desired time (with some probability).

The organization and contributions of this paper are as follows. Section 2 briefly reviews the syntax and semantics of APT-LPs from [19]. Section 3 defines the basic APT abduction problem and shows the complexity of checking for the existence of a solution (which is Σ_2^P-complete) and the complexity of checking whether a given set is a solution (which is D^P-complete). Section 4 provides a geometric intuition behind APT-LPs and derives conditions to prune the search space for a solution to BAAP. Section 5 derives a sound and complete algorithm for BAAP. Finally, Section 6 describes related work.

2 Preliminaries

We now briefly recall the syntax and semantics of APT-LPs from [19]. We assume the existence of a finite set $\mathcal{L}_{cons}$ of constant symbols, a finite set $\mathcal{L}_{pred}$ of predicate symbols,

[1] `http://www.aaas.org/news/releases/2011/0113rwanda.shtml?sa_campaign=Internal_Ads%2fAAAS%2fAAAS_News%2f2011-01-13%2fjump_page`, Jan. 13, 2011.

and an infinite set $\mathcal{L}_{var}$ of variable symbols. Each predicate symbol $p \in \mathcal{L}_{pred}$ has an *arity*. A *term* is any member of $\mathcal{L}_{cons} \cup \mathcal{L}_{var}$. If $t_1, \ldots, t_n$ are terms and $p \in \mathcal{L}_{pred}$ is a predicate symbol with arity n, then $p(t_1, \ldots, t_n)$ is an *atom*. Every atom is a formula. If F and G are formulas, then $F \wedge G$, $F \vee G$ and $\neg F$ are formulas.[2] A formula is *ground* iff no variables occur in it. $\mathcal{B}$ denotes the Herbrand base, i.e., the set of all ground atoms. We assume an arbitrarily large, but fixed size window of time $\tau = \{1, \ldots, t_{max}\}$.

▶ **Definition 1** (Annotated formula)**.** If F is a (ground) formula, $t \in \tau$ is a time point, and $[\ell, u] \subseteq [0, 1]$, then $F : [t, \ell, u]$ is an *(ground) annotated formula*.

Intuitively, $F : [t, \ell, u]$ says that F will be true at time t with a probability in the range $[\ell, u]$. We assume the existence of a finite set $\mathcal{F}$ of symbols called *frequency function symbols* denoting "frequency functions," which will be defined shortly.

▶ **Definition 2** (APT rule/program)**.** Let F and G be two (ground) formulas, Δt be a time interval, $[\ell, u] \subseteq [0, 1]$, $\mathsf{fr} \in \mathcal{F}$ be a frequency function symbol, and $[\alpha, \beta] \subseteq [0, 1]$.

1. $F \overset{\mathsf{fr}}{\rightsquigarrow} G : [\Delta t, \ell, u]$ is called an *(ground) unconstrained APT rule*.
2. $F \overset{\mathsf{fr}}{\hookrightarrow} G : [\Delta t, \ell, u, \alpha, \beta]$ is called a *(ground) constrained APT rule*.

An (ground) *APT program* is a finite set of (ground) APT rules and annotated formulas.

We now define the semantics of APT-programs.

▶ **Definition 3** (World)**.** A *world* is any set of ground atoms. We use $2^{\mathcal{B}}$ to denote the set of all worlds.

As a world is just an ordinary Herbrand interpretation, satisfaction of a formula F by a world w, denoted $w \models F$, is defined in the usual way [14].

▶ **Definition 4** (Thread)**.** A *thread* is a mapping $Th : \tau \to 2^{\mathcal{B}}$.

Intuitively, $Th(t)$ is the set of ground atoms which are true at time t according to Th. We use $\mathcal{T}$ to denote the set of all possible threads and now define a tp-interpretation.

▶ **Definition 5** (Temporal Probabilistic Interpretation)**.** A temporal probabilistic (tp) interpretation I is a probability distribution over the set of all possible threads, i.e., $I : \mathcal{T} \to [0, 1]$ and $\sum_{Th \in \mathcal{T}} I(Th) = 1$.

A tp-interpretation I assigns a probability to each thread.

▶ **Definition 6** (Satisfaction of an Annotated Formula)**.** Let $F : [t, \ell, u]$ be a ground annotated formula, and I be a tp-interpretation. We say that I *satisfies* $F : [t, \ell, u]$, denoted $I \models F : [t, \ell, u]$, iff $\ell \leq \sum_{Th \in \mathcal{T}, Th(t) \models F} I(Th) \leq u$.

A tp-interpretation satisfies an annotated formula iff it satisfies all its ground instances. Each frequency function symbol in an APT rule denotes a frequency function (FF) which tries to capture (within a thread) how often G is true Δt units after F. FFs are defined axiomatically by [19].

▶ **Definition 7** (Frequency Function)**.** Let Th be a thread, F and G be ground formulas, and $\Delta t > 0$ be an integer. A *frequency function* fr is a mapping of quadruples $(Th, F, G, \Delta t)$ to $[0, 1]$ such that:

[2] Throughout this paper, negation $\neg$ is treated in a classical manner and not as non-monotonic negation.

(FF1) If G is a tautology, then $\mathsf{fr}(Th, F, G, \Delta t) = 1$.
(FF2) If F is a tautology and G is a contradiction, then $\mathsf{fr}(Th, F, G, \Delta t) = 0$.
(FF3) If F is a contradiction, then $\mathsf{fr}(Th, F, G, \Delta t) = 1$.
(FF4) If G is not a tautology, and either F or $\neg G$ is not a tautology, and F is not a contradiction, then there exist threads $Th_1, Th_2 \in \mathcal{T}$ such that $\mathsf{fr}(Th_1, F, G, \Delta t) = 0$ and $\mathsf{fr}(Th_2, F, G, \Delta t) = 1$.

The rationale behind the above axioms is given in [19], which also introduces two useful FFs, the *point frequency function* (pfr) and the *existential frequency function* (efr):

$$pfr(Th, F, G, \Delta t) = \frac{|\{t : Th(t) \models F \wedge Th(t + \Delta t) \models G\}|}{|\{t : (t \leq t_{max} - \Delta t) \wedge Th(t) \models F\}|}$$

If there is no $t \in [0, t_{max} - \Delta t]$ such that $Th(t) \models F$ then we define pfr to be 1. Intuitively, pfr expresses how frequently G follows F in exactly Δt time points.

$$efr(Th, F, G, \Delta t) = \frac{efn(Th, F, G, \Delta t, 0, t_{max})}{|\{t : (t \leq t_{max} - \Delta t) \wedge Th(t) \models F\}| + efn(Th, F, G, \Delta t, t_{max} - \Delta t, t_{max})}$$

Here $efn(Th, F, G, \Delta t, t_1, t_2) = |\{t : (t_1 \leq t \leq t_2)$ and $Th(t) \models F$ and there exists $t' \in [t + 1, \min(t_2, t + \Delta t)]$ such that $Th(t') \models G\}|$. If the denominator is zero (if there is no $t \in [0, t_{max} - \Delta t]$ such that $Th(t) \models F$ and $efn(Th, F, G, \Delta t, t_{max} - \Delta t, t_{max}) = 0$) then we define efr to be 1. Intuitively, efr indicates the frequency that G follows F within Δt time points.

We now define satisfaction of APT rules by tp-interpretations.

▶ **Definition 8** (Satisfaction of APT rules). Let r be a ground APT rule with FF fr and I be a tp-interpretation.
1) If $r = F \overset{\mathsf{fr}}{\rightsquigarrow} G : [\Delta t, \ell, u]$, then we say that I *satisfies* r (denoted $I \models r$) iff $\ell \leq \sum_{Th \in \mathcal{T}} I(Th) \cdot \mathsf{fr}(Th, F, G, \Delta t) \leq u$.
2) If $r = F \overset{\mathsf{fr}}{\hookrightarrow} G : [\Delta t, \ell, u, \alpha, \beta]$, then we say that I *satisfies* r (denoted $I \models r$) iff $\ell \leq \sum_{Th \in \mathcal{T}, \alpha \leq \mathsf{fr}(Th, F, G, \Delta t) \leq \beta} I(Th) \leq u$.

Intuitively, the unconstrained APT rule $F \overset{\mathsf{fr}}{\rightsquigarrow} G : [\Delta t, \ell, u]$ evaluates the probability that F leads to G in Δt time units as follows: for each thread, it finds the probability of the thread according to I and then multiplies that by the frequency (in terms of fraction of times) with which F is followed by G in Δt time units according to frequency function fr. It then sums up these products across all threads in much the same way as an expected value computation. For constrained rules, the probability is computed by first finding all threads such that the frequency of F leading to G in Δt time units is in the $[\alpha, \beta]$ interval, and then summing up the probabilities of all such threads. This probability is the sum of probabilities assigned to threads where the frequency with which F leads to G in Δt time units is in $[\alpha, \beta]$. To satisfy the constrained APT rule $F \overset{\mathsf{fr}}{\hookrightarrow} G : [\Delta t, \ell, u, \alpha, \beta]$, this probability must be within the probability interval $[\ell, u]$. A tp-interpretation I satisfies an APT-rule iff it satisfies all its ground instances, and it satisfies an APT-program Π, denoted $I \models \Pi$, iff I satisfies all rules and annotated formulas in it.

An APT program Π is *consistent* iff there exists a tp-interpretation I s.t. $I \models \Pi$. Π *entails* an annotated formula f (resp., an APT rule r), denoted $\Pi \models f$ (resp., $\Pi \models r$), iff every tp-interpretation satisfying Π satisfies f (resp., r) as well.

3 Abduction in APT Logic

Given an APT program modeling a particular situation, we may want to know how to act to induce some goal condition at a given time in the future and within specified probability bounds.

▶ **Definition 9** (Basic APT Abduction Problem (BAAP))**.** An instance of the basic APT abduction problem is a triple $\langle \Pi, H, g \rangle$, where Π is an APT program, H is a finite set of annotated formulas and g is an annotated formula. $S \subseteq H$ is a solution to BAAP iff $\Pi \cup S$ is consistent and $\Pi \cup S \models g$.

Intuitively, Π models an environment, g is a goal we want to achieve, and H represents possibilities for what we can do in order to achieve the goal. Note that g is an annotated formula of the form $G : [t, \ell, u]$, where G is an *arbitrary* boolean combination.

▶ **Example 10.** Suppose we have an instance $\langle \Pi, H, g \rangle$ of BAAP, where Π is the simple APT program consisting of the following rules:

$$b \overset{pfr}{\rightsquigarrow} a : [1, 0.68, 0.82]$$
$$c \overset{pfr}{\rightsquigarrow} a : [1, 0.47, 0.7]$$

the set of abducibles is $H = \{b : [1, 0.3, 0.5], c : [1, 1, 1]\}$, and the goal is $g = a : [2, 0.6, 0.85]$. In addition, assume that for this problem $t_{max} = 2$. We must find a subset S of formulas from H such that $\Pi \cup S$ is consistent and entails g, that is, a solution where a will be true at time 2 with a probability in the range $[0.6, 0.85]$. Consider the subset $S = \{c : [1, 1, 1]\}$. When this formula is added to Π, we have that the resulting program $\Pi \cup \{c : [1, 1, 1]\}$ is consistent and entails $a : [2, 0.68, 0.7]$, which entails the goal formula $a : [2, 0.6, 0.85]$. Thus, S is a solution to this instance of BAAP.

Different preference criteria among solutions might be expressed in order to identify "preferred" solutions. Due to space constraints, in this paper we restrict ourselves to the Basic APT Abduction Problem defined above, thus no preference relation is considered.

Checking for the existence of a solution is Σ_2^P-complete even for restricted classes of APT programs.

▶ **Theorem 11** (BAAP Existence)**.** *Let $P = \langle \Pi, H, g \rangle$ be an instance of BAAP. Deciding whether a solution exists for P is Σ_2^P-complete. Σ_2^P-hardness holds even if Π is empty.*

Checking if a given set is a solution to BAAP is D^P-complete even for restricted classes of APT programs.

▶ **Theorem 12** (BAAP Checking)**.** *Let $P = \langle \Pi, H, g \rangle$ be an instance of BAAP. Deciding whether $S \subseteq H$ is a solution for P is D^P-complete. Hardness holds even if Π is restricted to be a program containing only annotated formulas, only unconstrained rules, or only constrained rules.*

It is easy to see that $\langle \Pi, H, g \rangle$ does not have solutions if at least one of Π and g is inconsistent; thus, in the rest of this paper we assume that both Π and g are consistent. A consistency checking algorithm is given in [19].

4 Geometric Characterization of APT Abduction

In this section, we present a geometric characterization of abduction in APT logic. As shown in [19], given an APT program Π, we can derive a linear program $\mathsf{LC}(\Pi)$ s.t. the solutions of $\mathsf{LC}(\Pi)$ are in $1-1$ correspondence with the interpretations satisfying Π. Clearly, $\mathsf{LC}(\Pi)$ defines a convex polytope denoted $Polytope(\Pi)$. For any two polytopes P_1 and P_2, we write $P_1 \subseteq P_2$ iff P_1 is contained in P_2, whereas $P_1 \cap P_2$ denotes the intersection of P_1 and P_2.

We can apply this geometric interpretation to BAAP as follows: given an instance $P = \langle \Pi, H, g\rangle$ of the basic APT abduction problem, $S \subseteq H$ is a solution of P iff $Polytope(\Pi \cup S) \neq \emptyset$ and $Polytope(\Pi \cup S) \subseteq Polytope(\{g\})$. Intuitively, we can think of BAAP as the problem of "cutting" $Polytope(\Pi)$ by using half-spaces in H so that we obtain a non-empty polytope which fits inside $Polytope(\{g\})$.

The following simple proposition provides a sufficient condition for the intersection of the polytope of an APT program and the polytope of an annotated formula to be empty. This proposition will effectively provide a pruning condition that we can exploit when looking for a solution.

▶ Proposition 4.1. Let Π be a consistent APT program and $F : [t, \ell, u]$ a consistent annotated formula. If $[\ell', u']$ is a probability interval s.t. $Polytope(\Pi) \subseteq Polytope(\{F : [t, \ell', u']\})$ and $[\ell', u'] \cap [\ell, u] = \emptyset$, then $Polytope(\{F : [t, \ell, u]\}) \cap Polytope(\Pi') = \emptyset$ for any $\Pi' \supseteq \Pi$.

By leveraging the geometric characterization of the APT abduction problem, we can make the following observations that play an important role in developing an algorithm for BAAP.

- If $Polytope(\Pi) \cap Polytope(\{g\}) = \emptyset$, then the problem does not have a solution—no matter how we slice $Polytope(\Pi)$, it will never fit inside $Polytope(\{g\})$. Alternatively, if $Polytope(\Pi) \subseteq Polytope(\{g\})$, then $\emptyset$ is a solution—we do not need to cut $Polytope(\Pi)$ as it already fits within $Polytope(\{g\})$, i.e., $\Pi \models g$.
- If neither of the above conditions holds, then we iterate over all possible subsets of H. There are two possible reasons why a subset S of H may *not* be a solution:
  1. $Polytope(\Pi \cup S) = \emptyset$, that is, $\Pi \cup S$ is inconsistent. In this case, any superset of S is also not a solution.
  2. $Polytope(\Pi \cup S) \neq \emptyset$, but $Polytope(\Pi \cup S) \not\subseteq Polytope(\{g\})$. In this case, any subset of S is also not a solution.

     In addition, Proposition 4.1 allows us to say even more. If $g = G : [t, \ell, u]$ and $[\ell', u']$ is a probability interval s.t. $Polytope(\Pi \cup S) \subseteq Polytope(\{G : [t, \ell', u']\})$ but $[\ell', u'] \cap [\ell, u] = \emptyset$, then $Polytope(\Pi \cup S)$ and $Polytope(\{g\})$ are not overlapping. Thus, any additional cuts to $\Pi \cup S$ would be useless, that is, any superset of S is also not a solution. Intuitively, this is the case where Π has been cut too much by S.

5 Basic APT Abduction Algorithm

We now present a sound and complete algorithm for BAAP. We note that the search space of possible solutions $\mathcal{H} = 2^H$ is a lattice under $\subseteq$. A brute-force algorithm to solve BAAP would traverse this lattice, inspecting one element at a time and checking whether or not that element is a solution. The algorithm would halt when it arrived at a solution (or when all lattice elements had been considered).

▶ **Example 13.** Suppose we have an arbitrary instance $\langle \Pi, H, g \rangle$ of BAAP, where $|H| = 4$. $\mathcal{H} = 2^H$ has 16 elements and is a complete lattice under the $\subseteq$ ordering. All 16 elements need to be considered, either implicitly or explicitly.

Instead of blindly choosing an element from this lattice, we can find a better way of choosing an element $S \in \mathcal{H}$ such that (i) S has a good chance of being a solution, and (ii) if it is not a solution, then we are able to prune the search space to avoid inspecting elements of $\mathcal{H}$ that are definitely not solutions.

As for the first point, we note that bigger subsets of H make more cuts to Π, but they also have a greater chance of being inconsistent with Π. For example, an extreme case would be the entire set H. If $\Pi \cup H$ is consistent, then we can determine whether or not the abduction problem has a solution without looking at any other subset of H; however, $\Pi \cup H$ has a high likelihood of being inconsistent. On the other hand, smaller subsets of H are more likely to be consistent, but we risk not cutting Π enough to entail the goal. For example, $\emptyset$ is consistent, but it is unlikely to be a solution unless $Polytope(\Pi) \subseteq Polytope(\{g\})$. Thus, a "medium-sized" set might be a good compromise between these two extremes.

Regarding the second point above, as already mentioned in the previous section, when we find that a subset $S \in \mathcal{H}$ is not a solution, we can prune the search space by discarding either all supersets or all subsets (or even both when Proposition 4.1 applies) of S. However, as the following example will demonstrate, the amount of potential pruning depends on the size of the subset chosen.

▶ **Example 14.** Consider again the instance of Example 13. Suppose we choose a "small" $S_1 \subseteq H$, e.g., where $|S_1| = 1$. If S_1 is not a solution because $Polytope(\Pi \cup S_1) \not\subseteq Polytope(\{g\})$, then we can discard only the empty set from the search space. On the other hand, suppose we choose a "big" $S_2 \subseteq H$, e.g., where $|S_2| = 3$. If S_2 is not a solution because $Polytope(\Pi \cup S_2) = \emptyset$, then we can discard only H from the search space. Note that if S_1 fails to be a solution because $Polytope(\Pi \cup S_1) = \emptyset$, or S_2 is not a solution because $Polytope(\Pi \cup S_2) \not\subseteq Polytope(\{g\})$, then the search space is pruned more, but these cases are less likely to occur. Thus, in the worst case we will end up pruning only one element. If we choose an $S \subseteq H$ having cardinality 2 and it fails to be a solution, then it is guaranteed that at least 3 elements from the search space will be discarded, regardless of why S is not a solution. A "medium-sized" set looks like a promising choice—because it is located in the center of the lattice surrounded by many elements, it can provide the largest amount of pruning in the worst case.

In the previous example, we saw that a subset S of H having cardinality 2 would be a good choice. In addition, proceeding by choosing medium-sized, centrally-located sets will move towards a solution in a binary search fashion, providing faster convergence. Identifying such a medium-sized set is easy in the first iteration, but as we move through the search space looking for a solution and pruning parts of the lattice, it becomes more difficult to determine how to choose a good element to inspect. Following the same intuitions introduced above, we would like to choose an element of the lattice from an area where there are many other elements, and in addition, choose one that has a medium size among those.

To identify the medium-sized subsets from a populous region of the search space, we use the scoring function defined below.

▶ **Definition 15.** Consider an instance $\langle \Pi, H, g \rangle$ of BAAP. Let $\mathcal{P} \subseteq 2^H$, $S \in \mathcal{P}$, $v_{max} = max\{|S'| \ : \ S' \in \mathcal{P} \wedge S \subseteq S'\}$, and $v_{min} = min\{|S'| \ : \ S' \in \mathcal{P} \wedge S' \subseteq S\}$. We define $score(S, \mathcal{P}) = (v_{max} - v_{min}) - abs(|S| - (v_{max} + v_{min})/2)$.

In the previous definition, $(v_{max} - v_{min})$ gives a measure of how many elements are around S whereas $abs(|S| - (v_{max} + v_{min})/2)$ gives a measure of the distance from S to a medium-sized element among those surrounding S. Note that in this definition, $\mathcal{P}$ is any subset of 2^H, and represents the remaining portions of the original search space $\mathcal{H}$ that have not yet been pruned.

▶ **Example 16.** Once again, consider the instance of Example 13 and let $\mathcal{P}$ be the whole search space $\mathcal{H} = 2^H$. For the top element of the lattice, $v_{max} = 4$, since the largest superset in $\mathcal{P}$ is H itself, and $v_{min} = 0$, since the smallest subset in $\mathcal{P}$ is the empty set. Using the scoring function from Definition 15, the score of the top element is $(4-0)-abs(4-(\frac{4+0}{2})) = 2$. Likewise, we can compute the scores of the remaining elements in the lattice: all sets of cardinality 3 have score 3, all sets of cardinality 2 have score 4, sets of cardinality 1 have score 3, and the score of the empty set is 2. As described in Example 14, the medium-sized elements of size 2 in the middle of the lattice are the most preferable choices for pruning, and have thus been assigned the highest score.

The observations made so far lead us to the algorithm reported below for solving the basic APT abduction problem.

Algorithm 1 Basic Abduction

Input: Instance $P = \langle \Pi, H, g = G : [t, \ell, u] \rangle$ of the basic APT abduction problem
Output: A solution for P if it exists, *false* otherwise

```
1: if Polytope(Π) ∩ Polytope({g}) = ∅ then
2:    return false
3: if Polytope(Π) ⊆ Polytope({g}) then
4:    return ∅
5: H := 2^H
6: while H ≠ ∅ do
7:    Choose S ∈ H having maximum score(S, H)
8:    [ℓ', u'] = Tightest(Π ∪ S, G, t)
9:    if [ℓ', u'] ∩ [ℓ, u] = ∅ then
10:      H := H − {S' | S' ∈ H ∧ S' ⊇ S}
11:   if [ℓ', u'] ≠ ∅ then
12:      if [ℓ', u'] ⊆ [ℓ, u] then
13:         return S
14:      else
15:         H := H − {S' | S' ∈ H ∧ S' ⊆ S}
16: return false
```

The algorithm first checks whether no solution exists because $Polytope(\Pi) \cap Polytope(\{g\}) = \emptyset$ or if the empty set is a trivial solution (lines 1–4). These checks can be done by solving a linear program as described in [19].

After that, in line 6 we begin to traverse the search space $\mathcal{H}$, pruning as the algorithm proceeds. In line 7 we choose a set $S \in \mathcal{H}$ with the maximum score. The function $Tightest(\Pi \cup S, G, t)$ in line 8 returns the tightest interval $[\ell', u']$ s.t. $Polytope(\Pi \cup S) \subseteq Polytope(\{G : [t, \ell', u']\})$ if $\Pi \cup S$ is consistent, otherwise it returns $\emptyset$. Again, [19] shows how this function can be computed by solving a linear program, providing different optimizations that can be applied to significantly reduce the size of these constraints. In lines 9 and 10 the search space is pruned by discarding every superset of S. Note that here we handle both the case where $\Pi \cup S$ is inconsistent and the case where $Polytope(\Pi \cup S)$ and $Polytope(\{g\})$ do not overlap, i.e., when Proposition 4.1 can be applied. If S is a solution, then it is returned

(lines 11–13). Line 15 examines the case where $\Pi \cup S$ is consistent but not a solution—here every subset of S is discarded. If no solution can be found, then the algorithm returns *false*.

▶ **Theorem 17.** *Let P be an instance of BAAP. If P has a solution, then Algorithm Basic Abduction returns one solution of P, otherwise the algorithm returns false.*

▶ **Example 18.** Suppose we have an instance $\langle \Pi, H, g \rangle$ of BAAP that we want to solve using Algorithm Basic Abduction, where Π is the simple APT program given in Example 10, $H = \{f_1 = b : [1, 0.4, 0.8], f_2 = c : [1, 0.8, 1.0], f_3 = c : [1, 1, 1], f_4 = b : [1, 1, 1]\}$, and $g = a : [2, 0.6, 0.85]$. First, we check whether the problem has no solution because $Polytope(\Pi) \cap Polytope(\{g\}) = \emptyset$, or a trivial solution if $Polytope(\Pi) \subseteq Polytope(\{g\})$. Since neither of these conditions are true, we begin searching $\mathcal{H}$ in line 6. As described in Example 16, in the initial lattice, elements of cardinality 2 have the maximum score of 4, so we choose the first such set $S = \{f_1, f_2\}$. Next, computing $Tightest(\Pi \cup S, G, t)$ on line 8 returns $[\ell', u'] = [0.68, 0.9]$. We now check to see if S is a solution, and if not, what elements we can prune from $\mathcal{H}$. Because $[\ell', u'] \cap [0.6, 0.85] \neq \emptyset$ on line 9, we cannot prune any supersets of S. However, because $[\ell', u'] \not\subseteq [0.6, 0.85]$ we can prune all subsets of S on line 14, removing the sets $\{f_1\}, \{f_2\}$, and $\emptyset$ from $\mathcal{H}$. We now begin the loop on line 6 again with our pruned $\mathcal{H}$. This time, the maximum score is 2.5 for subsets of cardinality 2 or 3, so we choose one of these elements—the next subset of cardinality 2—$S = \{f_1, f_3\}$. This time, $Tightest(\Pi \cup S, G, t)$ on line 8 returns $[\ell', u'] = [0.68, 0.7]$. Again, $[\ell', u'] \cap [0.6, 0.85] \neq \emptyset$ on line 9, so we continue to the next test on line 12. Since $[0.68, 0.7] \subseteq [0.6, 0.85]$, $Polytope(\Pi \cup S) \subseteq Polytope(\{g\})$, so we return $S = \{b : [1, 0.4, 0.8], c : [1, 1, 1]\}$ as a BAAP solution.

6 Related Work and Conclusion

Though abduction has been extensively studied [3, 7, 16, 8], there is no work that we are aware of that studies abduction in the context of both probabilities and time.

There has been important work on abduction in temporal logic. [2] presents a non-deterministic algorithm to find explanations for temporal phenomena based on a framework called STP. [5] is another important paper that uses constraint checking methods and compilation methods to achieve greater efficiency. [6] develops an SLDNF-based procedure to perform temporal abduction in logic programs based on the Abductive Event Calculus [6, 9, 21, 12] using a notion of partial plans. [1] provides an abduction mechanism that allows the definition of preferences over abductive explanations. However, none of these frameworks involves uncertainty.

Abduction has also been studied in the context of uncertainty. However, all past work on abduction in such settings has been devised under various independence assumptions [18, 17, 4]. The only exceptions are [24, 23] which perform abduction in possible worlds-based probabilistic logic systems such as those of [11], [15], and [10] where independence assumptions are not made. In this paper, no independence assumption is made, and moreover, none of the above frameworks include time in addition to probabilities. The first implementation of abductive logic programs without independence assumptions is contained in [23, 22].

In this paper, we have shown how abduction can be performed in the context of logic programs containing both probabilistic and temporal information. We have formally defined the Basic APT Abduction Problem (BAAP). We have shown that the problem of determining existence of a solution to BAAP is Σ_2^P-complete whereas checking if a given set is a solution to BAAP is D^P-complete. We have developed strategies to prune the search space for a solution and have given an algorithm to compute a solution to BAAP.

Much work remains to be done. It is only recently that practical algorithms to compute consistency and entailment in logic programs with probability and time have been developed based on randomized algorithms [20]. Likewise, it is only recently that algorithms to compute abductive explanations to probabilistic logic programs *without independence assumptions* have been devised [23, 22]. Scalable implementations of algorithms for BAAP are necessary. We are developing randomized and sampling-based algorithms for BAAP in the spirit of [13] which scale well to very large numbers of possible worlds and threads. We are also developing an application of APT Abduction to the problems described in this paper.

References

1 C. Baral. Abductive reasoning through filtering. *Artif. Intell.*, 120(1):1–28, June 2000.
2 V. Brusoni, L. Console, P. Terenziani, and D. Theseider Dupré. An efficient algorithm for temporal abduction. *AI*IA*, pages 195–206, 1997.
3 T. Bylander, D. Allemang, M. C. Tanner, and J. R. Josephson. The computational complexity of abduction. *Artif. Intell.*, 49((1–3)):25–60, 1991.
4 H. Christiansen. Implementing probabilistic abductive logic programming with constraint handling rules. In *Constraint Handling Rules*, pages 85–118. 2008.
5 T. Console, P. Terenziani, and D.T. Dupré. Local reasoning and knowledge compilation for temporal abduction. *IEEE Trans. Knowl. Data Eng.*, 14(6):1230–1248, 2002.
6 M. Denecker, L. Missiaen, and M. Bruynooghe. Temporal reasoning with abductive event calculus. In *In Proc. of the European Conference on Artificial Intelligence*, pages 384–388. John Wiley & Sons, 1992.
7 T. Eiter and G. Gottlob. The complexity of logic-based abduction. *Journal of the ACM*, 42(1):3–42, 1995.
8 T. Eiter, G. Gottlob, and N. Leone. Semantics and complexity of abduction from default theories. *Artif. Intell.*, 90:90–1, 1997.
9 U. Endress, P. Mancarella, F. Sadri, G. Terreni, and F. Toni. The ciff proof procedure for abductive logic programming with constraints. *Logics in Artificial Intelligence*, 3229:31–43, 2004.
10 R. Fagin, J. Y. Halpern, and N. Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87(1/2):78–128, 1990.
11 T. Hailperin. Probability logic. *Notre Dame Journal of Formal Logic*, 25 (3):198–212, 1984.
12 A. Kakas, R. Kowalski, and F. Toni. Abductive logic programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.
13 S. Khuller, M. V. Martinez, D. S. Nau, A. Sliva, G. I. Simari, and V. S. Subrahmanian. Computing most probable worlds of action probabilistic logic programs: scalable estimation for $10^{30,000}$ worlds. *Ann. Math. Artif. Intell.*, 51(2-4):295–331, 2007.
14 J. W. Lloyd. *Foundations of Logic Programming.* Springer, 2nd edition, 1987.
15 N. Nilsson. Probabilistic logic. *Artif. Intell.*, 28:71–87, 1986.
16 Y. Peng and J. Reggia. *Abductive Inference Models for Diagnostic Problem Solving.* Springer-Verlag, 1990.
17 D. Poole. Probabilistic horn abduction and bayesian networks. *Artif. Intell.*, 64(1):81–129, 1993.
18 D. Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artif. Intell.*, 94(1-2):7–56, 1997.
19 P. Shakarian, A. Parker, G. I. Simari, and V. S. Subrahmanian. Annotated probabilistic temporal logic. *ACM Trans. Comput. Log.*, 12(2), 2011.
20 P. Shakarian, G.I. Simari, and V.S. Subrahmanian. Annotated probabilistic temporal logic: Approximate fixpoint implementation. *ACM Trans. Comput. Log.*, to appear, 2011.

21 M. Shanahan. An abductive event calculus. *Journal of Logic Programming*, 44(1–3):207–240, 2000.

22 G. I. Simari, J. Dickerson, A. Sliva, and V.S. Subrahmanian. Parallel abductive query answering in probabilistic logic programs. *submitted to a technical journal*, Dec. 2010.

23 G. I. Simari, J. P. Dickerson, and V. S. Subrahmanian. Cost-based query answering in action probabilistic logic programs. In *SUM*, pages 319–332, 2010.

24 G. I. Simari and V. S. Subrahmanian. Abductive inference in probabilistic logic programs. In *ICLP'10 (Technical Communications)*, volume 7 of *LIPIcs*, pages 192–201. Schloss Dagstuhl, 2010.