Statistical abduction with tabulation[1]

Taisuke SATO[†] and Yoshitaka KAMEYA[‡]

Dept. of Computer Science, Graduate School of Information
Science and Engineering, Tokyo Institute of Technology
2-12-1 Ookayama Meguro-ku Tokyo Japan 152-8552
(sato@mi.cs.titech.ac.jp[†], kame@mi.cs.titech.ac.jp[‡])

Abstract.

We propose *statistical abduction* as a first-order logical framework for representing, inferring and learning probabilistic knowledge. It semantically integrates logical abduction with a parameterized distribution over abducibles. We show that statistical abduction combined with tabulated search provides an efficient algorithm for probability computation, a Viterbi-like algorithm for finding the most likely explanation, and an EM learning algorithm (the graphical EM algorithm) for learning parameters associated with the distribution which achieve the same computational complexity as those specialized algorithms for HMMs (hidden Markov models), PCFGs (probabilistic context-free grammars) and sc-BNs (singly connected Bayesian networks).

1 Introduction

Abduction is a form of inference that generates the best explanation for observed facts. For example, if one notices that the grass is wet in the yard, he/she might abduce that it rained last night, or the sprinkler was on, by using general rules such as "if it rains, things get wet." Abduction has been used for diagnosis systems [30], planning [14, 41], natural language processing [5, 15], user modeling [9] etc in AI.

It is possible to formalize (part of) abduction in logic programming as follows [16, 17]. We have a background theory T consisting of clauses and an observed fact G (usually a ground atom) to be explained, and the task of abduction is to search for an explanation $E = \{a_1, \ldots, a_n\}$ by choosing ground atoms a_is from a particular class of primitive hypotheses called *abducibles*[2] such that $T \cup E \models G$ and $T \cup E$ is consistent.[3] The quality of E, the abduced explanation, is evaluated by various criteria such as precision, simplicity, abduction cost, and so on [15, 16, 40].

[1] This paper is based on a workshop paper presented at the UAI-2000 workshop on Fusion of Domain Knowledge with Data for Decision Support, Stanford, 2000.

[2] The term "explanation" is henceforth used as a synonym of a conjunction (or set) of abducibles.

[3] Sometimes $T \cup E$ is required to satisfy integrity constraints, but in this paper, we do not consider them.

While the above framework is simple and logically sound, it is obviously incomplete. Especially it entirely ignores the problem of uncertainty in the real world. Our observations are often partial, inconsistent or contaminated by noise. So the abduced explanation should be treated as being true only to some degree. Also it must be noticed that our observations are always finite but potentially infinite (we may have another observation indefinitely), and it is often critical to evaluate how far our explanation holds on average. Since these problems are certainly not in the realm of logic, but belong to statistics, it is natural and desirable to build an interdisciplinary framework that unifies the logical aspects and the statistical aspects of abduction.

There are many ways of doing this, but one of the simplest ways is to introduce a *parameterized probability distribution over abducibles*. We term the resulting logical-statistical framework *statistical abduction*, in which we calculate the probabilities of explanations from the parameters associated with the distribution and determine the most likely explanation among possible ones as the one with the highest probability. Parameters are statistically learnable from observations if they are unknown.

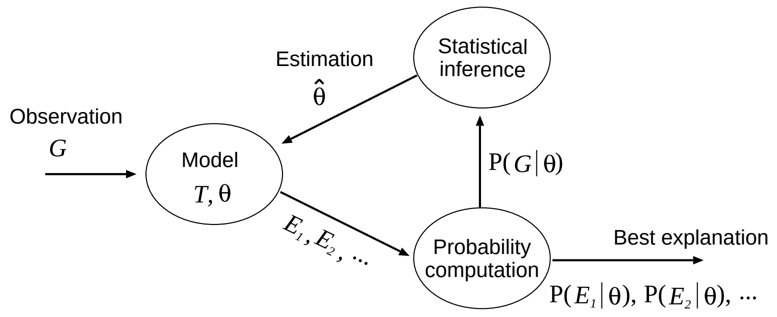

Figure 1: Statistical abduction

The idea of statistical abduction is illustrated above. We have abducibles $a_1, a_2, \ldots$ with a probability distribution parameterized by θ, and a clausal theory T. For a given G, observation, we search for possible explanations $E_1, E_2, \ldots$ each of which is a conjunction of finitely many abducibles. $P(E_1 \mid \theta), P(E_2 \mid \theta), \ldots$, probabilities of explanations, are computed from marginal distributions for these constituent abducibles. Their probabilities are used to select the best explanation and also to compute $P(G \mid \theta)$. The parameter θ is estimated by applying ML (maximum likelihood) estimation to $P(G \mid \theta)$.

We would like to first emphasize that statistical abduction is not merely abduction of logical explanations but aims at the inference of their distribution. Second, it has wide coverage, as we will see, ranging from logic programming to popular symbolic-statistical frameworks. By popular symbolic-statistical frameworks, we mean for instance HMMs (hidden Markov Models), PCFGs (probabilistic context free grammars) and BNs (Bayesian networks) explained below.

An HMM is a special type of Markov chain in which a symbol is emitted at a state and we can only observe a sequence of emitted symbols whereas state transitions are not observable, i.e. they are hidden. HMMs are used as a modeling tool for speech recognition, genome informatics etc [22, 32]. Also a PCFG is a context free grammar with probabilities assigned to each rule in such a way that if there are n production rules $A \rightarrow B_1, \ldots, A \rightarrow B_n$ for a non-terminal symbol A, probabilities p_i is assigned to $A \rightarrow B_i$ $(1 \leq i \leq n)$ where $\sum_i p_i = 1$. The probability of a parse tree is the product of probabilities assigned to rules appearing in the tree, and that of a sentence is the sum of probabilities of possible parse trees for the sentence [6, 22, 43]. PCFGs form a basis of stochastic natural language processing. Finally, a Bayesian network means an acyclic directed graph consisting of nodes of random variables where a child node conditionally depends on the parent nodes, and the dependency is specified by a CPT (conditional probability table) when nodes take discrete values. BNs are used to model probabilistic-causal relationships. A *singly connected* BN is one that does not include loops when directions are ignored in the graph [4, 28].

Turning back to statistical abduction, we note that all statistical techniques from fitting test to random sampling and to parameter learning are applicable. They provide us with powerful means for the statistical analysis of abduction logically formalized.

There are however two fundamental problems; one is theoretical and the other is practical. First of all, statistical abduction must deal with infinitely many objects sanctioned by the language of first-order logic and their joint distributions, which raises the mathematical question of defining a probability space consistently giving a joint distribution over a set of arbitrarily chosen objects. It goes beyond probabilistic semantics often seen in AI that deals only with finite domains and finitely many random variables.

Secondly, to apply it in practice, we need to know *all* values of statistical parameters, but determining a large number of statistical parameters is a hard task, known as the where-do-the-numbers-come-from problem. Although one might hope that the problem is mitigated by learning, there has been little work on parameter learning in the literature of logical framework of abduction.

The objective of this paper is to make it clear that there exists a firm theoretical basis for statistical abduction and we have an efficient algorithm for computing probabilities, thereby being able to efficiently determine the most likely hypothesis and an efficient EM algorithm for parameter learning.[4] Since the subject is broad and the space is limited, we concentrate on putting the major ideas across and details are left to the related literature [18, 33, 34, 35, 36, 37].[5] In what follows, after reviewing some historical background in Section 2, we sketch our probabilistic semantics in Section 3, and explain in Section 4 PRISM, a symbolic-statistical modeling language implementing distribution semantics as an embodiment of statistical abduction. Section 5 is a main section. We first

[4] The EM algorithm is an iterative algorithm which is a standard method for ML estimation of statistical parameters from incomplete data [23].

[5] We submitted a comprehensive paper on the subject [38].

describe three basic computational tasks required for statistical abduction. We then propose the use of tabulated search and combine it with general algorithms for PRISM programs to perform the three tasks, and finally state the time complexity of PRISM programs for the case of HMMs, PCFGs and sc-BNs (singly connected Bayesian networks), which indicates that the proposed algorithms run as efficiently as specialized algorithms for HMMs, PCFGs and sc-BNs. Section 6 is a conclusion.

2 Background

Looking back on the role of probabilities in logic programming, two approaches, "constraint approach" and "distribution approach," are distinguishable. The former focuses on the inference of probability intervals assigned to atoms and clauses as constraints, whereas the prime interest of the latter is to represent a single probability distribution over atoms and formulas, from which various statistics are calculated.

The constraint approach is seen in the early work of Nilsson [27] where he tried to compute, by using linear programming technique, the upper and lower bound of probability of a target sentence consistent with a first-order knowledge base in which each sentence is assigned a probability. In logic programming, Ng and Subrahmanian took the constraint approach to formulate their probabilistic logic programming [25] (see [12] for recent development). Their program is a set of annotated clauses of the form $A : \mu_0 \leftarrow F_1 : \mu_1, \ldots, F_n : \mu_n$ where A is an atom, F_i $(1 \leq i \leq n)$ a basic formula, i.e. a conjunction or a disjunction of atoms, and the annotation μ_j $(0 \leq j \leq n)$ a sub-interval in $[0, 1]$ indicating a probability range. A query $\leftarrow \exists(F_1 : \mu_1, \ldots, F_n : \mu_n)$ is answered by an extension of SLD refutation. Their language contains only a finite number of constant and predicate symbols, and no function symbols are allowed.

A similar probabilistic framework was proposed by Lakshmanan and Sadri under the same syntactic restrictions (finitely many constant, predicate symbols, no function symbols) in a different uncertainty setting [21]. They used annotated clauses of the form $A \overset{c}{\leftarrow} B_1, \ldots, B_n$ where A and B_i are atoms and $c = \langle [\alpha, \beta], [\gamma, \delta] \rangle$, the confidence level, represents the belief interval $[\alpha, \beta]$ $(0 \leq \alpha \leq \beta \leq 1)$ and doubt interval $[\gamma, \delta]$ $(0 \leq \gamma \leq \delta \leq 1)$ which an expert has in the clause [21].

By comparison, the distribution approach has been actively pursued outside logic programming. In particular, researchers in the Bayesian network community have been using definite clauses with probabilities attached to express probabilistic events such as gene inheritance. In the framework of KBMC (knowledge-based model construction) [1, 3, 19, 26] for instance, clauses are used as a macro language to compactly represent similar Bayesian networks. Basically a knowledge base KB contains clauses representing general rules and CPTs (conditional probability tables). Every time a set of evidence and context is given as ground atoms, a specialized Bayesian network is constructed by tracing logical/probabilistic dependencies in KB to compute the probability of a query atom. Uncertain parameters associated with CPTs can be learned by applying the EM learning algorithm for Bayesian networks [4] to the constructed network [19]. It can be said that KBMC implicitly defines a collection of local distributions in the form of Bayesian networks, each corresponding to a pair of evidence and context. The question of whether there exists a single distribution compatible with these implicitly defined (and infinitely many) local distributions or not remains open.

In contrast to KBMC, statistical abduction explicitly defines a single distribution (probability measure) over ground atoms. It was begun by Poole as "probabilistic Horn abduction" [31]. In his approach, a program is comprised of non-probabilistic definite clauses and probabilistic disjoint declarations. A disjoint declaration is of the form `disjoint([`h_1`:`p_1`,...,`h_n`:`p_n`])`. It says h_i, an abducible atom, becomes exclusively true with probability p_i $(1 \leq i \leq n)$. Abducibles in different declarations are independent. The probability of a non-abducible ground atom is then calculated by reduction in a top-down manner through program clauses to a DNF formula made out of abducibles in the disjoint declarations. The probabilistic Horn abduction is able to represent Bayesian networks [31].

While the probabilistic Horn abduction opened a new vista on extending Bayesian networks to first-order languages, it makes various assumptions on programs such as the acyclicity condition[6] and the covering property.[7] These assumptions are not easy to verify and could be severe restrictions in programming. For example, under the acyclicity condition, when a clause includes local variables like Y in $p(X) \leftarrow q(X,Y), \ldots$ one cannot write recursive clauses about q such as $member(X, cons(H,Y)) \leftarrow member(X,Y)$. Also the defined probability measure is not proved to be completely additive either. In other words, the continuity $\lim_{n\rightarrow\infty} P(p(t_1) \vee \ldots \vee p(t_n)) = P(\exists X p(X))$ where t_is are ground terms, is not necessarily guaranteed. More serious is the problem of determining parameters in disjoint declarations. How can we get them? It remained unanswered.

SLP (stochastic logic programming) proposed by Muggleton [24] is another attempt to define probabilities over ground atoms. He associated, analogously to PCFGs, probabilities p_i's with range-restricted clauses[8] C_i's like $p_i : C_i$ ($1 \leq i \leq n$). The probability of a ground atom G is defined as the product of such p_is appearing in G's SLD refutation, but with a modification such that if a subgoal g can invoke n clauses, $p_i : C_i$ ($1 \leq i \leq n$) at some derivation step, the probability of choosing k th clause is normalized to $p_k / \sum_{i=1}^{n} p_i$. More recently, Cussens extended SLP by introducing the notion of loglinear models for SLD refutations and defined probabilities of ground atoms in terms of their SLD-

[6]It says that every ground atom A must be assigned a unique integer $n(A)$ such that $n(A) > n(B_1), \ldots, n(B_n)$ for every ground instance of a clause of the form $A \leftarrow B_1, \ldots, B_n$.

[7]It requires that when there are finite ground instances $A \leftarrow \alpha_i$ $(1 \leq i \leq m)$ about a ground atom A in the program, $A \leftrightarrow \alpha_1 \vee \ldots \vee \alpha_m$ holds. Intuitively the property ensures every observation has an explanation. Logically it is equivalent to assuming the iff completion [13].

[8]A clause is range-restricted if variables appearing in the head also appear in the body. So, a unit clause must be ground.

trees and "features" [10]. To define the probability of a ground atom $s(a)$, he first defines the probability $P(R)$ of an SLD refutation R for the most general goal $\leftarrow s(X)$ as $P(R) \stackrel{\mathrm{def}}{=} Z^{-1}\exp\left(\sum_i \log(\lambda_i) f(R,i)\right)$. Here λ_i is a number (parameter) associated with a clause C_i and $f(R,i)$ is a feature such as the number of occurrences of C_i in R. Z is a normalizing constant. The probability assigned to a ground atom $s(a)$ is the sum of probabilities of all possible SLD refutations for $\leftarrow s(a)$ [10]. An EM algorithm for inferring parameters taking failures into account is proposed in [11]. Presently, assigning probabilities to arbitrary quantified formulas is out of the scope of both of SLPs.

Looking at the distribution approach to probabilistic functional languages, we notice that Koller et al. proposed a probabilistic functional language which can represent HMMs, PCFGs and BNs [20], but neither the problem of defining declarative semantics nor that of learning parameters in a program was not discussed. Later Pfeffer developed it into another functional language with declarative semantics which is based on the products of countably infinite uniform distributions over the unit interval $[0, 1)$. EM learning is sketched [29].

3 Distribution semantics: an overview

Aiming at providing a broader theoretical basis and a learning algorithm for statistical parameters of statistical abduction, Sato proposed *distribution semantics* [33] and developed a first-order statistical modeling language PRISM (`http://mi.cs.titech.ac.jp/prism/`) [33, 34]. The proposed semantics rigorously defines a *probability measure* over the set of Herbrand interpretations as the denotation of a PRISM program. It is exactly a probabilistic extension of the least Herbrand model semantics to the possible world semantics with a probability measure, but eliminates extraneous assumptions made in the previous approaches. For example, there is no need for the covering assumption or the acyclicity condition [31] (because every definite program has a least Herbrand model and the iff completion [13] holds in it.). Similarly, neither the range-restrictedness condition nor normalization in SLPs [10, 24] is necessary. What is more, there is no vocabulary restriction. We may use as many constant symbols, function symbols and predicate symbols as we need, and can write whatever program we want, though in actual programming, we have to care about efficiency, termination, etc.

Syntactically, our program DB is a set $F \cup R$ where F is a set of atoms (abducibles) and R is a set of definite clauses such that no clause head in R is unifiable with an atom in F. In the theoretical context however, we always consider DB as a set of ground clauses made up of all possible ground instances of the original clauses in DB. F then is a set of infinitely many ground atoms. We associate with F a *basic probability measure* P_F. It is defined over the set of Herbrand interpretations of F and makes every atom A in F a random variable taking on 1 when A is true and 0 otherwise. Hence atoms in F are probabilistically true and random sampling determines a set of true atoms F'. Then think of a new definite clause program $DB' = F' \cup R$ and its least Herbrand model $M(DB')$ [13]. $M(DB')$ determines the truth values of all ground atoms in DB, which implies that every ground atom in DB is a random variable. Therefore a probability measure P_{DB} over the set of Herbrand interpretations of DB is definable [33]. P_F mentioned above is constructed from a collection of finite joint distributions $P_F^{(n)}(A_1 = x_1, \ldots, A_n = x_n)$ $(n = 1, 2, \ldots)$ where A_is $(\subset F)$ are random variables (abducibles) such that

$$P_F^{(n)}(A_1 = x_1, \ldots, A_n = x_n) = \sum_{x_{n+1} \in \{0,1\}} P_F^{(n+1)}(A_1 = x_1, \ldots, A_{n+1} = x_{n+1}).$$

In the following, for the sake of intuitiveness, we use a joint distribution and a probability measure interchangeably. This is because the probability measure P_F behaves as if it were an infinite probability distribution whose marginal distribution is $P_F^{(n)}(\cdot)$ $(n = 1, 2, \ldots)$.[9]

PRISM, an implementation of the distribution semantics with P_F chosen to be a specific form (the direct products of infinitely many random switches) has been developed as a symbolic-statistical modeling language for complex phenomena governed by rules and probabilities [33, 34, 35]. It is a general logic programming language equipped with a built-in EM algorithm by which we can learn parameters associated with P_F from observations represented by ground atoms. As PRISM allows us to use programs to specify distributions (*programs as distributions*), we have an enormous degree of freedom and flexibilities in modeling complex symbolic-statistical phenomena. Actually, we have found it rather easy to write a PRISM program modeling complicated interactions between gene inheritance and social rules (bi-lateral cross cousin marriage) observed in the Kariera tribe, an anthropological tribe which lived 80 years ago in the west Australia [35, 44].

4 PRISM programs

In this section, we explain PRISM programs by examples. In our framework, observations are represented by ground atoms and the role of PRISM programs is to specify their joint distributions in terms of built-in probabilistic atoms (abducibles).

A PRISM program is a definite clause program $DB = F \cup R$ such that R, a set of definite clauses, represents non-probabilistic rules such as Mendel's law whereas F, a set of ground atoms, represents basic probabilistic events and has an infinite joint distribution P_F. F and P_F must satisfy the following.

1. F is a set of probabilistic atoms of the form `msw(`i`,`n`,`v`)`. They are random variables taking on 1 (resp. 0) when true (resp. false). The arguments i and n are ground terms called *switch name* and *trial-id*, respectively. We assume that V_i, a finite set of ground terms, is associated with each i, and $v \in V_i$ holds. V_i is called the *value set* of i.

[9] This also applies to P_{DB}.

2. Let V_i be $\{v_1, v_2, \ldots, v_{|V_i|}\}$. Then, one of the ground atoms msw(i,n,v_1), msw(i,n,v_2), ..., msw(i,n,$v_{|V_i|}$) becomes exclusively true on each trial. For each i, $\theta_{i,v} \in [0,1]$ is a *parameter* of the probability of msw(i,$\cdot$,v) being true ($v \in V_i$), and $\sum_{v \in V_i} \theta_{i,v} = 1$ holds.

3. For arbitrary i, i', n, n', $v \in V_i$ and $v' \in V_{i'}$, random variable msw(i,n,v) is independent of msw(i',n',v') if $n \neq n'$ or $i \neq i'$.

A ground atom msw(i,n,v) represents an event "a probabilistic switch named i takes on v as a sample value on the trial n" (msw stands for *multi-valued switch*). The second and the third condition say that a logical variable V in msw(i,n,V) behaves like a random variable which is realized to v_k with probability θ_{i,v_k} ($k = 1 \ldots |V_i|$). Moreover, from the third condition, the logical variables V1 and V2 in msw(i,n_1,V1) and msw(i,n_2,V2) can be seen as *independent and identically distributed* (i.i.d.) random variables if n_1 and n_2 are different ground terms. From an abductive point of view, msw atoms are abducibles.[10]

To get a feel for PRISM programs, we first take a look at a non-recursive PRISM program. Imagine a lawn beside a road and their observations such as "the road is dry but the lawn is wet." Assume that the lawn is watered by a sprinkler that (probabilistically) works only when it does not rain. The process that generates an observation observed(road(X),lawn(Y)) ("the road is X and the lawn is Y") where X,Y $\in$ {wet,dry} is described by the program $DB_{\texttt{rs}}$ in Figure 2.

```
(1) target(observed/2).
(2) values(rain,[yes,no]).
(3) values(sprinkler,[on,off]).
(4) observed(road(X),lawn(Y)):-
       msw(rain,once,A),
       ( A = yes, X = wet, Y = wet
       ; A = no,  msw(sprinkler,once,B),
           ( B = on,  X = dry, Y = wet
           ; B = off, X = dry, Y = dry ) ).
```

Figure 2: $DB_{\texttt{rs}}$

This program first declares observed/2 as a target predicate corresponding to our observations by clause (1). (2) and (3) declare the use and value sets of msw atoms. For example (2) declares a probabilistic multi-ary switch named rain whose values are {yes,no}. (4), the main clause defining observed/2 is read like an ordinary Prolog clause. The difference between (4) and usual clauses is two usages of built-in msw atoms in the body. msw(rain,once,A) for example returns in A one of {yes,no} sampled according to a parameterized

[10] The second and the third condition correspond to the disjoint declaration in Poole's framework [31]: disjoint([msw(i,N,v_1):θ_{i,v_1}, ..., msw(i,N,$v_{|V_i|}$):$\theta_{i,v_{|V_i|}}$]).

distribution $P_{F_{\mathtt{r}}}(\cdot \mid \theta_{\mathtt{r}})$ described below. `msw(sprinkler,once,B)` behaves similarly.[11] If disjunctions look messy, by the way, it is possible to split the clause into three clauses each of which has a conjunctive body. By doing so however, we will have a multiple occurrences of the same `msw` atom.

Write the program as $DB_{\mathtt{rs}} = F_{\mathtt{rs}} \cup R_{\mathtt{rs}}$ where $F_{\mathtt{rs}} = \{\mathtt{msw(rain, once, yes)}, \mathtt{msw(rain, once, no)}, \mathtt{msw(sprinkler, once, on)}, \mathtt{msw(sprinkler, once, off)}\}$ and $R_{\mathtt{rs}}$ is the set of ground instantiations of (4). To define a basic distribution $P_{F_{\mathtt{rs}}}$ over $F_{\mathtt{rs}}$, put $F_{\mathtt{r}} = \{\mathtt{msw(rain, once, yes)}, \mathtt{msw(rain, once, no)}\}$ and introduce a distribution $P_{F_{\mathtt{r}}}(\cdot,\cdot)$ over $F_{\mathtt{r}}$ parameterized by $\theta_{\mathtt{r}}$ $(0 \leq \theta_{\mathtt{r}} \leq 1)$ such that[12]

$$\begin{aligned}
P_{F_{\mathtt{r}}}(\mathtt{msw(rain, once, yes)} = 1, \mathtt{msw(rain, once, no)} = 1) &= 0 \\
P_{F_{\mathtt{r}}}(\mathtt{msw(rain, once, yes)} = 1, \mathtt{msw(rain, once, no)} = 0) &= \theta_{\mathtt{r}} \\
P_{F_{\mathtt{r}}}(\mathtt{msw(rain, once, yes)} = 0, \mathtt{msw(rain, once, no)} = 1) &= 1 - \theta_{\mathtt{r}} \\
P_{F_{\mathtt{r}}}(\mathtt{msw(rain, once, yes)} = 0, \mathtt{msw(rain, once, no)} = 0) &= 0.
\end{aligned}$$

Introduce analogously another distribution $P_{F_{\mathtt{s}}}(\cdot,\cdot)$ parameterized by $\theta_{\mathtt{s}}$ over the set $F_{\mathtt{s}} = \{\mathtt{msw(sprinkler, once, on)}, \mathtt{msw(sprinkler, once, off)}\}$. The basic distribution $P_{F_{\mathtt{rs}}}$ is then defined as the products of $P_{F_{\mathtt{r}}}$ and $P_{F_{\mathtt{s}}}$. Hereafter for simplicity, we use $P(A)$ as a synonym for $P(A = 1)$, and $P(\neg A)$ for $P(A = 0)$. Accordingly we write $P_{DB_{\mathtt{rs}}}(\mathtt{observed(road(dry), lawn(wet))}) = (1 - \theta_{\mathtt{r}})\theta_{\mathtt{s}}$ etc.

PRISM provides the user with not only various built-ins to set statistical parameter values and compute probabilities of atoms using them, but a built-in EM learning routine for ML (maximum likelihood) estimation to infer parameter values from observed atoms. That is if we have a random sample such as

$$\mathtt{observed(road(wet), lawn(wet))}, \mathtt{observed(road(dry), lawn(wet))}, \ldots$$

we can statistically infer $\theta_{\mathtt{r}}$ and $\theta_{\mathtt{s}}$ from them as the maximizers of the likelihood of the sample (we further discuss EM learning later).

Now we turn to another feature of PRISM, recursion. The existence of recursion in a program potentially introduces a countably infinite number of random variables and the construction of an underlying probability space is an absolute necessity for their joint distributions to be consistently defined, but presents some technical difficulties. Distribution semantics however achieves it through the least model semantics [33].

As an example of recursive PRISM program, we look at an HMM program $DB_{\mathtt{hmm}}$ in Figure 4 describing a two state HMM in Figure 3 that generates strings $\{\mathtt{a}, \mathtt{b}\}^*$ (of finite length, 3 in this case). In the program, clause (1) declares that only ground atoms containing `hmm/1` are observable. (2) is concerned with tabulated search which will be explained later. Since `msw` atoms that can appear as goals during execution have similar patterns, (4) and (5) declare them by

[11] If a ground `msw` atom such as `msw(rain,once,yes)` is called, we first execute `msw(rain,once,A)` and then execute `A = yes`. So the goal fails if the sampled value returned in `A` is `no`.

[12] "`once`" in `msw(rain,once,yes)` is a constant to identify a trial that is attempted only once in the program.

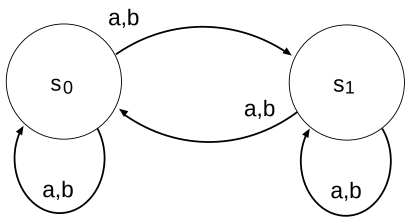

Figure 3: Two state HMM

```
(1) target(hmm/1).
(2) table([hmm/1,hmm/3]).
(3) values(init,[s0,s1]).
(4) values(out(_),[a,b]).
(5) values(tr(_),[s0,s1]).
(6) hmm(Cs):- msw(init,once,Si),hmm(1,Si,Cs).
(7) hmm(T,S,[C|Cs]):- T=<3,
        msw(out(S),T,C),msw(tr(S),T,NextS),
        T1 is T+1,hmm(T1,NextS,Cs).
(8) hmm(T,_,[]):- T>3.
```

Figure 4: PRISM program DB_{hmm} for the two state HMM

terms containing "_" that matches anything. Clauses (6)$\sim$(8) specify the probabilistic behavior of the HMM. `T` is a time step and `S` and `NextS` are states. `Cs` represents a list of output symbols. Clause (7) probabilistically chooses an output symbol `C` and the next state `NextS`. To represent switches sampled at each state `S`, it uses non-ground terms `out(S)` and `tr(S)`. `T` is used to guarantee independence among choices at different time steps. DB_{hmm} as a whole describes a process of stochastic generation of strings such as `hmm([a,b,a])`. The program is procedurally understandable by Prolog reading except `msw` atoms. That is, given ground `S` and `T`, `C` in `msw(out(S),T,C)` behaves like a random variable taking discrete values {`a`,`b`} declared by clause (4).

5 Three computational tasks

To apply statistical abduction to the real world, we need computational tractability in addition to expressive power. We here consider three basic computational tasks based on the analogy of HMMs [32]:

(1) computing $P_{DB}(G \mid \vec{\theta})$,[13] the probability of an atom G representing an observation,

[13] $\vec{\theta}$ is the vector consisting of parameters associated with all abducibles which forms the explanations for the observed fact G or an observation in $\mathcal{G}$.

(2) finding E^*, the most likely explanation for G, and

(3) adjusting the parameters so that the probability of a given sequence $\mathcal{G} = \langle G_1, G_2, \ldots, G_T \rangle$ of observations is maximized.

All solutions should be computationally tractable.

As for HMMs, these methods correspond to the forward procedure, the Viterbi algorithm and the Baum-Welch algorithm respectively [32, 22].

Poole [31] described a method for the first task. Let us consider a program $DB = F \cup R$ and the following if-and-only-if (iff) relation under $comp(R)$, the Clark's completion of the rules R [7]:

$$comp(R) \models G \leftrightarrow E^{(1)} \vee \cdots \vee E^{(m)}. \tag{1}$$

He assumes that there exist finitely many explanations $\psi_{DB}(G) = \{E^{(1)}, \cdots, E^{(m)}\}$ for G as above each of which is a finite conjunction of independent abducibles and they are mutually exclusive, i.e. $P_{DB}(E^{(i)} \wedge E^{(j)}) = 0$ $(1 \leq i \neq j \leq m)$ (we say DB satisfies the *exclusiveness condition*). Let I be the set of switch names and $\sigma_{i,v}(E)$ the number of occurrences of $\texttt{msw}(i,\cdot,v)$ in an explanation E. His method for solving the first task is formulated in our notation as follows:

$$P_{DB}(G \mid \vec{\theta}) = \textstyle\sum_{E \in \psi_{DB}(G)} P_F(E) = \sum_{E \in \psi_{DB}(G)} \prod_{i \in I, v \in V_i} \theta_{i,v}^{\sigma_{i,v}(E)}. \tag{2}$$

A little modification of the above formula would give one for the second task:

$$E^* = \arg\max_{E \in \psi_{DB}(G)} \textstyle\prod_{i \in I, v \in V_i} \theta_{i,v}^{\sigma_{i,v}(E)}. \tag{3}$$

Unfortunately, $|\psi_{DB}(G)|$, the number of explanations for G, often grows exponentially in the complexity of the model (e.g. the number of states in an HMM), or in the complexity of each observation (e.g. the string length).

5.1 OLDT search and support graphs

For the three basic computational tasks to be practically achievable, it is a must to suppress computational explosions. Define anew $\psi_{DB}(G)$, the set of all explanations for a goal G, by $\psi_{DB}(G) \overset{\text{def}}{=} \{E \mid \text{minimal } E \subset F, R \cup E \vdash G\}$.

Statistical abduction has two potential sources of computational explosions. One is a search phase searching for $\psi_{DB}(G)$. It will be explosive if the search is done by backtracking as can be easily confirmed by the HMM program $DB_{\texttt{hmm}}$. The other is a probability computation phase corresponding to (2) and/or (3). They would be explosive without factoring computations. Suppose we have a program $\{$`g:-m1,m2,m3. g:-m4,m2,m3.`$\}$ in which `g` is a goal. (2) leads us to $P(\texttt{g}) = P(\texttt{m1})P(\texttt{m2})P(\texttt{m3}) + P(\texttt{m4})P(\texttt{m2})P(\texttt{m3})$. However this computation repeats the same computation $P(\texttt{m2})P(\texttt{m3})$ twice.

It is possible to avoid these computational redundancies all at once by adopting tabulated search that results in a compact graphical representation of all solutions, which in turn makes it possible to factor probability computations. The point is to introduce intermediate atoms between a goal and abducibles called *table atom*s to factor out common probability computations and let them store the computed results. In the above case, we should write {`g:-m1,h. g:-m4,h. h:-m2,m3.`} using a new table atom `h`. We compute $P(\texttt{h}) = P(\texttt{m2})P(\texttt{m3})$ once and use the result twice later in the computation of $P(\texttt{g}) = P(\texttt{m1})P(\texttt{h}) + P(\texttt{m4})P(\texttt{h})$. The remaining of this subsection and the next subsection detail the idea sketched above.

In OLDT search [42] which is a complete tabulated search method for logic programs that adopts the tabling of goals, we store explanations for a goal G in a global table called a *solution table* while letting them hierarchically share common sub-explanations [18, 42]. Such hierarchical sharing reflects on factoring probability computations carried out after search. From the solution table, a graph called *support graph* representing $\psi_{DB}(G)$ is extracted as an ordered set of disconnected subgraphs. Once the support graph is extracted, it is relatively easy to derive algorithms for the three computational tasks that run as efficiently as specialized ones such as the forward procedure, the Viterbi algorithm and the Baum-Welch algorithm in HMMs, as we see later.

Mathematically we need some assumptions to validate the derivation of these efficient algorithms. Namely we assume that the number of explanations for a goal is finite (we say DB satisfies the *finite support condition*) and there exists a linearly ordered set[14] $\tau_{DB}(G) \overset{\text{def}}{=} \langle \tau_0, \tau_1, \ldots, \tau_K \rangle$ ($\tau_0 = G$) of *table atoms*[15] satisfying the following conditions:

- Under *comp*(R), a table atom τ_k ($0 \leq k \leq K$) is equivalent to a disjunction $E_{k,1} \vee \cdots \vee E_{k,m_k}$. Each disjunct $E_{k,h}$ ($1 \leq h \leq m_k$) is called a *tabled-explanation* for τ_k and made up of `msw` atoms and other table atoms. The set $\tilde{\psi}_{DB}(\tau_k) = \{E_{k,1}, \ldots, E_{k,m_k}\}$ is called the *tabled-explanation*s for τ_k. Logically, it must hold that

$$\begin{aligned} comp(R) \models\ & (G \leftrightarrow E_{0,1} \vee \cdots \vee E_{0,m_0}) \\ & \wedge (\tau_1 \leftrightarrow E_{1,1} \vee \cdots \vee E_{1,m_1}) \\ & \wedge \cdots \wedge (\tau_K \leftrightarrow E_{K,1} \vee \cdots \vee E_{K,m_K}). \end{aligned} \tag{4}$$

  Also we require that table atoms be layered in the sense that atoms appearing in the right hand side of $\tau_k \leftrightarrow E_{k,1} \vee \cdots \vee E_{k,m_k}$ belong in $F \cup \{\tau_{k+1}, \ldots, \tau_K\}$ (*acyclic support condition*). In other words, τ_k can only refer to $\tau_{k'}$ such that $k < k'$ in the program.

[14] Here we use a vector notation to emphasize the set is ordered.

[15] *Table atom*s mean atoms containing a *table predicate*. They make an entry for a table to store search results. Table predicates are assumed to be declared by the programmer in advance like `table([hmm/1,hmm/3])` in $DB_{\texttt{hmm}}$. We treat the top goal G as a special table atom τ_0.

```
(T1)  top_hmm(Cs,Ans):- tab_hmm(Cs,Ans,[]).
(T2)  tab_hmm(Cs,[hmm(Cs)|X],X):- hmm(Cs,Ans,[]).
(T2') tab_hmm(T,S,Cs,[hmm(T,S,Cs)|X],X):- hmm(T,S,Cs,Ans,[]).
(T3)  e_msw(init,T,s0,[msw(init,T,s0)|X],X).
(T3') e_msw(init,T,s1,[msw(init,T,s1)|X],X).
   :
(T6)  hmm(Cs,X0,X1):-
           e_msw(init,once,Si,X0,X2),
           tab_hmm(1,Si,Cs,X2,X1).
(T7)  hmm(T,S,[C|Cs],X0,X1):- T=<3,
           e_msw(out(S),T,C,X0,X2),
           e_msw(tr(S),T,NextS,X2,X3),
           T1 is T+1, tab_hmm(T1,NextS,Cs,X3,X1).
(T8)  hmm(T,S,[],X,X):- T>3.
```

Figure 5: Translated program DB^t_{hmm}

- $P_{DB}(E_{k,i}, E_{k,j}) = 0$ if $i \neq j$ for $E_{k,i}, E_{k,j} \in \tilde{\psi}_{DB}(\tau_k)$ $(0 \leq k \leq K)$ (*t-exclusiveness condition*) and each tabled-explanation in $\tilde{\psi}_{DB}(\tau_k)$ is comprised of statistically independent atoms (*independent condition*).

$\tau_{DB}(G)$ satisfying these conditions (the finite support condition, the acyclic support condition, the t-exclusiveness condition and the independent condition) is obtained, assuming due care is taken by the programmer, by (a specialization of) OLDT search [42] as follows. We look at the HMM program DB_{hmm} in Section 4 as a running example. We first translate DB_{hmm} to a Prolog program similarly to DCGs (*definite clause grammars*). The Prolog program DB^t_{hmm} in Figure 5 is a translation of DB_{hmm}. Clauses `(T`j`)` and `(T`j`')` are generated from the clause `(`j`)` in DB_{hmm}. In translation, we add two arguments as difference-list to atoms to hold a tabled-explanation. Table predicates do not change, so table predicates in DB^t_{hmm} are `hmm/3` and `hmm/5`. We rename the abducibles declared by `(3)`~`(5)` in DB_{hmm} so that they are placed in the callee's difference-list. We treat table atoms just like `msw` atoms except that they invoke subsequent calls to search for their tabled-explanations returned in `Ans`. For this purpose, we add clauses with the head of the form `tab_...()` like `(T2)`.

After translation, we fix the search strategy of OLDT to *multi-stage depth-first strategy* [42] (it is like Prolog execution) and run DB^t_{hmm} for a top-goal, for instance `:- top_hmm([a,b,a],Ans,[])` to search for all tabled-explanations of the tabled atom τ_0 = `hmm([a,b,a])` corresponding to our observation. They are returned in `Ans` as answer substitutions.

The top goal invokes a table atom `hmm([a,b,a],Ans,[])` through clause `(T2)`. Generally in OLDT search, when a table atom `hmm(`t`,Ans,[])` is called for the first time, we create a new entry `hmm(`t`)` in the solution table. Every time `hmm(`t`,Ans,[])` is solved with answer substitutions $t = t'$ and `Ans` $= e$, we store e (= a tabled-explanation for `hmm(`t'`)`) in the solution table under the sub-entry `hmm(`t'`)` (in the current case however, t and t' are ground, so they coincide). If the entry `hmm(`t`)` already exists, `hmm(`t`,Ans,[])` returns with one of the unused solutions. The OLDT search terminates exhausting all solutions for `hmm([a,b,a],Ans,[])` and yields a solution table in Figure 6.

```
hmm([a,b,a]):
    [hmm([a,b,a]): [[msw(init,once,s0),hmm(1,s0,[a,b,a])],
                    [msw(init,once,s1),hmm(1,s1,[a,b,a])]]]
hmm(1,s0,[a,b,a]):
    [hmm(1,s0,[a,b,a]):[[msw(out(s0),1,a),msw(tr(s0),1,s0),hmm(2,s0,[b,a])],
                        [msw(out(s0),1,a),msw(tr(s0),1,s1),hmm(2,s1,[b,a])]]]
hmm(1,s1,[a,b,a]):
    [hmm(1,s1,[a,b,a]):[[msw(out(s1),1,a),msw(tr(s1),1,s0),hmm(2,s0,[b,a])],
                        [msw(out(s1),1,a),msw(tr(s1),1,s1),hmm(2,s1,[b,a])]]]
 :
```

Figure 6: Part of solution table for `hmm([a,b,a])`.

A list

```
[[msw(init,once,s0),hmm(1,s0,[a,b,a])],
 [msw(init,once,s1),hmm(1,s1,[a,b,a])]]
```

under the sub-entry `hmm([a,b,a])` in Figure 6 means

$$\begin{aligned} \mathit{comp}(R_{\texttt{hmm}}) \quad \vdash \quad & \texttt{hmm}([\texttt{a},\texttt{b},\texttt{a}]) \leftrightarrow \\ & \quad (\texttt{msw}(\texttt{init},\texttt{once},\texttt{s0}) \wedge \texttt{hmm}(\texttt{1},\texttt{s0},[\texttt{a},\texttt{b},\texttt{a}]) \vee \\ & \quad (\texttt{msw}(\texttt{init},\texttt{once},\texttt{s1}) \wedge \texttt{hmm}(\texttt{1},\texttt{s1},[\texttt{a},\texttt{b},\texttt{a}]) \end{aligned}$$

where $R_{\texttt{hmm}} = \{$`(6)`, `(7)`, `(8)`$\}$ in $DB_{\texttt{hmm}}$ in Section 4.

After OLDT search, we collect all tabled-explanations from the solution table and topologically sort them to get linearly ordered table atoms $\tau_{DB}(G)$ $= \langle \tau_0, \tau_1, \ldots, \tau_K \rangle$ ($\tau_0 = G$) together with their tabled-explanations $\tilde{\psi}_{DB}(\tau_k)$ $(0 \leq k \leq K)$ satisfying Equation (4).

Since all the data we need in the subsequent computation of $P_{DB}(G \mid \vec{\theta})$ is $\tau_{DB}(G)$ (and $\tilde{\psi}_{DB}(\cdot)$), and since it is much more natural from a computational point of view to look upon $\tau_{DB}(G)$ as a graph than a set of atoms logically connected, we introduce a graphical representation of $\tau_{DB}(G)$, and call it a *support graph*. Namely, a *support graph* $\tau_{DB}(G)$ for G is a *linearly ordered* set $\langle \tau_0, \tau_1, \ldots, \tau_K \rangle$ ($\tau_0 = G$) of disconnected subgraphs. Each subgraph τ_k $(0 \leq k \leq K)$ (we identify a subgraph with the table atom labeling it) is comprised of linear graphs of the form `start-`$\mathtt{e}_1$`-`$\cdots$`-`$\mathtt{e}_M$`-end` representing some tabled-explanation $\mathtt{e}_1 \wedge \cdots \wedge \mathtt{e}_M$ for τ_k. Here `start` is a fork node and `end` is a join node and $\mathtt{e}_h$ $(1 \leq h \leq M)$ is either a `msw` atom or a table atom labeling the corresponding subgraph in a lower layer. Part of the support graph for $\tau_{DB}(\texttt{hmm}([\texttt{a},\texttt{b},\texttt{a}]))$ is described in Figure 7.

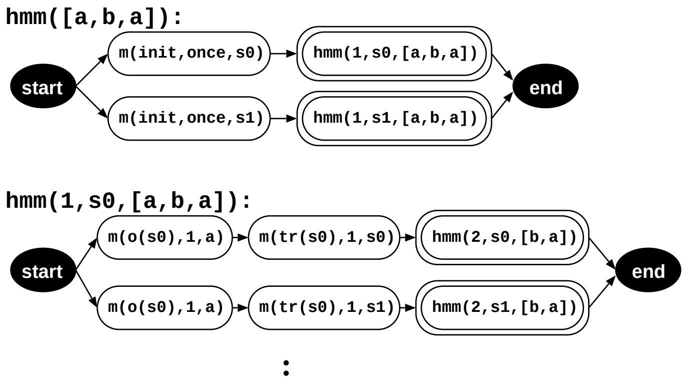

Figure 7: Part of the support graph for `hmm([a,b,a])`

5.2 Computing the observation probability and the most likely explanation

Given a support graph for G, an efficient algorithm for computing $P_{DB}(G \mid \vec{\theta})$ (the first task) is derived based on the analogy of the inside probabilities in Baker's Inside-Outside algorithm [2]. In our formulation, the inside probability of a table atom τ (sometimes called the *generalized inside probabilitiy* of τ) is $P_{DB}(\tau \mid \vec{\theta})$. Recall that the computation of $P_{DB}(G \mid \vec{\theta})$ by Equation (2) completely ignores the fact that $P_F(E)$ and $P_F(E')$ $(E \neq E')$ may have common computations, and hence always takes time proportional to the number of explanations in $\psi_{DB}(G)$.

The use of the support graph $\tau_{DB}(G) = \langle \tau_0, \tau_1, \ldots, \tau_K \rangle$ $(\tau_0 = G)$ enables us to factor out common computations. First note that distribution semantics ensures that $P_{DB}(\tau_k) = \sum_{E_{k,h} \in \tilde{\psi}_{DB}(\tau_k)} P_{DB}(E_{k,h})$ holds for every k $(0 \leq k \leq K)$ [33, 38]. Consequently from the support graph

$$\tau_{DB_{\text{hmm}}}(\texttt{hmm([a,b,a])}) = \langle \texttt{hmm([a,b,a])}, \texttt{hmm(1,s0,[a,b,a])}, \ldots, \texttt{hmm(4,s1,[])} \rangle$$

in Figure 7, we have

$$\left\{ \begin{array}{l} P(\texttt{hmm([a,b,a])}) \\ \qquad = \theta_{(\texttt{init},\texttt{s0})} P(\texttt{hmm(1,s0,[a,b,a])}) + \theta_{(\texttt{init},\texttt{s1})} P(\texttt{hmm(1,s1,[a,b,a])}) \\ P(\texttt{hmm(1,s0,[a,b,a])}) \\ \qquad = \theta_{(\texttt{out(s0)},\texttt{a})} \theta_{(\texttt{tr(s0)},\texttt{s0})} P(\texttt{hmm(2,s0,[b,a])}) \\ \qquad\quad + \theta_{(\texttt{out(s0)},\texttt{a})} \theta_{(\texttt{tr(s0)},\texttt{s1})} P(\texttt{hmm(2,s1,[b,a])}) \\ \quad \cdots \\ P(\texttt{hmm(4,s1,[])}) = 1 \end{array} \right.$$

Here $P(\cdot) = P_{DB_{\text{hmm}}}(\cdot)$. Second note that by computing inside probabilities sequentially from the bottom table atom `hmm(4,s1,[])` to the top table atom `hmm([a,b,a])`, we can obtain $P(\texttt{hmm}(1, \texttt{s1}, [\texttt{a}, \texttt{b}, \texttt{a}]))$ in time proportional to the size of the support graph which is $O(N^2L)$, not the number of all explanations $O(N^L)$, where N is the number of states and L the length of an input string.

The program Get-Inside-Probs below generalizes this observation. It takes as input a support graph $\tau_{DB}(G) = \langle \tau_0, \tau_1, \ldots, \tau_K \rangle$ ($\tau_0 = G$) for a goal G and computes inside probabilities $P_{DB}(\tau_k \mid \vec{\theta})$ starting from the bottom atom τ_K whose tabled explanations only contain `msw` atoms and ending at the top goal $\tau_0 = G$. In the program, a tabled explanation $E \in \tilde{\psi}_{DB}(\tau_k)$ is considered as a set and $\mathcal{P}[\cdot]$ is an array storing inside probabilities. It should be noted that, when computing $\mathcal{P}[\tau_k]$, the inside probabilities $\mathcal{P}[\tau_K]$, $\mathcal{P}[\tau_{K-1}]$, ..., $\mathcal{P}[\tau_{k+1}]$ have already been computed. The computation terminates leaving the inside probability of G in $\mathcal{P}[\tau_0](= \mathcal{P}[G])$.

```
1: procedure Get-Inside-Probs($\tau_{DB}(G)$) begin
2:   for $k := K$ downto 0 do
3:     $\mathcal{P}[\tau_k] := \sum_{E \in \tilde{\psi}_{DB}(\tau_k)} \prod_{\mathtt{msw}(i,\cdot,v) \in E} \theta_{i,v} \prod_{\tau \in E \cap \{\tau_{k+1},\ldots,\tau_K\}} \mathcal{P}[\tau]$
4: end.
```

Similarly, an efficient algorithm for computing the most likely explanation E^* for G (i.e. the second task) is derived. The algorithm Get-ML-Expl below first computes $\delta[\tau_k]$, the maximum probability of tabled-explanations for each table atom $\tau_k \in \tau_{DB}(G)$. $\mathcal{E}[\tau_k]$, the most likely tabled-explanation for τ_k, is simultaneously constructed. Finally, we construct E^* from $\mathcal{E}[\cdot]$.

```
1:  procedure Get-ML-Expl($\tau_{DB}(G)$) begin
2:    for $k := K$ downto 0 do begin
3:      foreach $E \in \tilde{\psi}_{DB}(\tau_k)$ do
4:        $\delta'[\tau_k, E] :=$
5:          $\prod_{\mathtt{msw}(i,\cdot,v) \in E} \theta_{i,v} \prod_{\tau \in E \cap \{\tau_{k+1},\ldots,\tau_K\}} \delta[\tau]$;
6:      $\delta[\tau_k] := \max_{E \in \tilde{\psi}_{DB}(\tau_k)} \delta'[\tau_k, E]$;
7:      $\mathcal{E}[\tau_k] := \arg\max_{E \in \tilde{\psi}_{DB}(\tau_k)} \delta'[\tau_k, E]$
8:    end;
9:    $s := \{G\}$;  $E^* := \emptyset$;
10:   while $s \neq \emptyset$ do begin
11:     Select and remove $A$ from $s$;
12:     if $A = \mathtt{msw}(\cdot,\cdot,\cdot)$ then add $A$ to $E^*$
13:     else $s := s \cup \mathcal{E}[A]$
14:   end
15: end
```

We remark that Get-ML-Expl is equivalent to the Viterbi algorithm, a standard algorithm for finding the most likely state-transition path of HMMs [22, 32]. Furthermore, in case of PCFGs (probabilistic context-free grammars), it is easily shown that a probabilistic parser for a PCFG in PRISM combined with Get-ML-Expl can find the most likely parse of a given sentence.

5.3 Graphical EM algorithm

The third task, i.e. the parameter learning of PRISM programs means ML estimation of statistical parameters $\vec{\theta}$ associated with `msw`s in a program DB, for which the EM algorithm is appropriate [23]. A new EM algorithm named *graphical EM algorithm* that runs on support graphs has been derived based on the analogy of computation of the *outside probabilities* in the Inside-Outside algorithm [18, 37]. We added an assumption that all observable atoms are exclusive to each other and their probabilities sum up to one to guarantee the mathematical correctness of the algorithm (we say DB satisfies the *uniqueness condition*).

Although details of the graphical EM algorithm are left to [18, 37], we give a brief account. The algorithm takes as input a set of support graphs $\{\tau_{DB}(G_1)$ $,\ldots,\tau_{DB}(G_T)\}$ generated from a random sample of goals $\langle G_1,\ldots,G_T\rangle$. It first initializes the parameters $\vec{\theta}$ randomly and then iterates the simultaneous update of $\vec{\theta}$ by executing the E(expectation) step followed by the M(aximization) step until the likelihood of the observed goals saturates. Final values $\vec{\theta}^*$ become the learned ones that locally maximize $\prod_{t=1}^{T} P_{DB}(G_t \mid \vec{\theta})$. The crux in the graphical EM algorithm is the E step, i.e. the computation of the expected counts of occurrences of `msw(`i`,`$\cdot$`,`v`)` in a proof of the top goal G:

$$\sum_{E\in\psi_{DB}(G)} \sigma_{i,v}(E)P_F(E \mid G,\vec{\theta})$$

to update a parameter $\theta_{i,v}$ associated with `msw(`i`,`$\cdot$`,`v`)`. Naive computation of the expected counts as above causes computation time to become proportional to the number of explanations for G, which must be avoided. We compute it indirectly using the *generalized outside probabilities* which are recursively (and efficiently) computed from the support graph $\tau_{DB}(G) = \langle\tau_0,\tau_1,\ldots,\tau_K\rangle$ for G like the inside probabilities but from τ_0 to τ_K. Update of parameters per iteration completes by scanning the support graph twice, once for inside probabilities and once for outside probabilities, and hence update time is linear in the size of $\tau_{DB}(G)$ [18, 37].

5.4 Complexity

In this subsection, we first analyze the time complexity of our methods for the first and the second task. The method for the first (resp. the second) task comprises two phases – OLDT search to generate a support graph for a goal and a subsequent computation by GET-INSIDE-PROBS (resp. by GET-ML-EXPL). Hence, we should estimate each phase separately. First assuming that table access can be performed in $O(1)$ time,[16] the computation time of OLDT search is measured by the size of the search tree which depends on a class of models.

[16]In reality $O(1)$ can be subtle. The worst case should be $O(\log n)$ for n data by using balanced trees [8]. In the case of PCFGs, n is N^3L^3 with N non-terminals for a sentence of length L, so $O(\log n) = O(\log \max\{N, L\})$. On the other hand, if we consider the abundance of memory available nowadays, it seems technically and economically reasonable to employ an

As for the computation time of GET-INSIDE-PROBS and GET-ML-EXPL, since both algorithms scan a support graph $\tau_{DB}(G)$ only once, it is linear in the size of $\tau_{DB}(G)$, or $O(\xi_{\mathrm{num}}\xi_{\mathrm{maxsize}})$ in notation where $\xi_{\mathrm{num}} \stackrel{\mathrm{def}}{=} |\Delta|$, $\xi_{\mathrm{maxsize}} \stackrel{\mathrm{def}}{=} \max_{E \in \Delta} |E|$, and $\Delta \stackrel{\mathrm{def}}{=} \bigcup_{\tau \in \tau_{DB}(G)} \tilde{\psi}_{DB}(\tau)$.

Now we examine concrete models. For an HMM program like $DB_{\mathtt{hmm}}$ in Section 4, OLDT time and the size of a support graph are both $O(N^2L)$ where N is the number of states and L the length of an input string. This is because for a ground top-goal $\mathtt{hmm}([w_1, \ldots, w_L])$ (we are thinking of $DB_{\mathtt{hmm}}$), there are at most NL goal patterns of table atom $\mathtt{hmm}(t,s,l)$ during the execution. Each goal causes N recursive calls in the body of clause (7) in $DB_{\mathtt{hmm}}$. Thanks to OLDT search, each table atom is computed once (we assume in the programming, the arguments t and s in $\mathtt{hmm}(t,s,l)$ are numbers and $O(1)$ table access is available). Therefore the size of the search tree is $O(N^2L)$ and so is the search time for all solutions. Also as each tabled-explanation is a conjunction of at most three atoms (see Figure 7), we conclude that $\xi_{\mathrm{num}} = O(N^2L)$ and $\xi_{\mathrm{maxsize}} = O(1)$. Hence, the time complexity of GET-INSIDE-PROBS and GET-ML-EXPL for HMMs becomes $O(N^2L)$. This is the same order as that of the forward procedure and the Viterbi algorithm. So GET-INSIDE-PROBS (resp. GET-ML-EXPL) is a generalization of the forward procedure (resp. the Viterbi algorithm).

For PCFGs, we assume grammars are in Chomsky normal form. Then it is shown that the time complexity of OLDT search is $O(M^3L^3)$ and so is the size of a support graph (see [18]), and hence GET-INSIDE-PROBS and GET-ML-EXPL run in time $O(M^3L^3)$. Here M is the number of non-terminals in the grammar, L the length of an input sentence.

Compared to HMMs and PCFGs, Bayesian networks present harder problems as computing marginal probabilities in a Bayesian network is NP-hard. So we focus on the sub-class, singly connected Bayesian networks [4, 28], though expressing general Bayesian networks by PRISM programs is straightforward [34]. By writing an appropriate PRISM program for a singly connected Bayesian network which has a clause corresponding to each node in the singly connected network, it is relatively easy to show that OLDT time for GET-INSIDE-PROBS and GET-ML-EXPL is linear in the number of nodes in the network [38]. We here assumed that the maximum number of parent nodes is fixed.

Our method for the third task (EM learning) comprises OLDT search and the graphical EM algorithm. For the latter, time complexity is measured by the re-estimation time per iteration (since we do not know how many times it iterates until convergence in advance). It is shown, analogously to GET-INSIDE-PROBS and GET-ML-EXPL however, to be $O(\xi_{\mathrm{num}}\xi_{\mathrm{maxsize}})$ for one goal. The reader is referred to [18, 37, 38] for details.

array in order to ensure $O(1)$ data access time, as has been traditionally assumed in parsing algorithms. Also we note that hashing achieves average $O(1)$ data access time under a certain assumption [8].

Model	OLDT time	GIP/GMLE	GEM
HMMs	$O(N^2L)$	$O(N^2L)$	$O(N^2LT)$
PCFGs	$O(M^3L^3)$	$O(M^3L^3)$	$O(M^3L^3T)$
sc-BNs	$O(\|V\|)$	$O(\|V\|)$	$O(\|V\|T)$

Table 1: Time complexity for the three computational tasks

We summarize computation time w.r.t. popular symbolic-statistical models in Table 1. In the table, the second column"OLDT time" indicates that computation time of OLDT search (assuming $O(1)$ table access) for the models in the first column. The third column "GIP/GMLE" means the time complexity of Get-Inside-Probs (the first task) and Get-ML-Expl (the second task) respectively corresponding to the model in the first column. The fourth column "GEM" is the time complexity of (one iteration of) the graphical EM algorithm (the third task, parameter estimation by EM learning). N, L, M, $|V|$ and T are respectively the number of states of the target HMM, the maximum length of input strings, the number of non-terminal symbols in the target PCFG, the number of nodes in the target singly connected Bayesian network and the size of training data. In statistical abduction with OLDT search, time complexity for each of the three computational tasks is the sum of OLDT time and the subsequent probability computations which is linear in the total size of support graphs.

Table 1 exemplifies that our general framework can subsume specific algorithms in terms of time complexity. For HMMs, $O(N^2L)$ is the time complexity of the forward algorithm, the Viterbi algorithm and one iteration of the Baum-Welch algorithm [32, 22]. $O(M^3L^3)$ is the time complexity of one iteration of the Inside-outside algorithm for PCFGs [2, 22]. $O(|V|)$ is the time complexity of a standard algorithm for computing marginal probabilities in singly connected Bayesian networks [28, 4] and that of one iteration of the EM algorithm for singly connected Bayesian networks [4].

6 Conclusion

We have proposed statistical abduction as a combination of abductive logic programming and a distribution over abducibles. It has first-order expressive power and integrates current most powerful probabilistic knowledge representation frameworks such as HMMs, PCFGs and (singly connected) Bayesian networks. Besides, thanks to a new data structure, support graphs which are generated from OLDT search, our general algorithms developed for the three computational tasks (probability computation, the search for the most likely explanation, and EM learning) accomplish the same efficiency as specialized algorithms for above three frameworks. On top of that, recent learning experiments with PCFGs by the graphical EM algorithm using two Japanese corpora of moderate size[17] suggest that the graphical EM algorithm can run much (orders of magnitude) faster than the Inside-Outside algorithm [37, 39].

There remains however a lot to be done including finishing the implementation of OLDT search and the proposed algorithms in PRISM and the development of various applications of statistical abduction. Also a theoretical extension to programs containing negation is an important research topic.

References

[1] Bacchus, F., Using First-Order Probability Logic for the Construction of Bayesian Networks, Proc. of UAI'93, pp219-226, 1993.

[2] Baker,J.K., Trainable Grammars for Speech Recognition, Proc. of Spring Conference of the Acoustical Society of America, pp547-550, 1979.

[3] Breese,J.S., Construction of Belief and Decision Networks, J. of Computational Intelligence, Vol.8 No.4 pp624-647, 1992.

[4] Castillo,E., Gutierrez,J.M., and Hadi,A.S., Expert Systems and Probabilistic Network Models, Springer-Verlag, 1997.

[5] Charniak,E., A neat theory of marker passing, Proc. of AAAI'86, pp584-588, 1986.

[6] Charniak,E., Statistical Language Learning, The MIT Press, 1993.

[7] Clark, K., Negation as failure, In Gallaire, H., and Minker, J. (eds), Logic and Databases, pp293-322, Plenum Press, 1978.

[8] Cormen,T.H., Leiserson,C. E. and Rivest,R.L., Introduction to Algorithms, MIT Press, 1990.

[9] Csinger,A., Booth,K.S. and Poole,D., AI Meets Authoring: User Models for Intelligent Multimedia, Artificial Intelligence Review 8, pp447-468, 1995.

[10] Cussens,J., Loglinear models for first-order probabilistic reasoning, Proc. of UAI'99, pp126-133, 1999.

[11] Cussens,J., Parameter estimation in stochasitc logic programs, Machine Learning 44, pp245-271, 2001.

[12] Dekhtyar,A.and Subrahmanian,V.S., Hybrid Probabilistic Programs, Proc. of ICLP'97, pp391-405, 1997.

[13] Doets,K., From Logic to Logic Programming, MIT Press, Cambridge, 1994.

[17] One corpus contains 9,900 sentences. It has a very ambiguous grammar (2,687 rules), generating 3.0×10^8 parses/sentence at the average sentence length 20. The other corpus consists of 10,995 sentences, and has a much less ambiguous grammar (860 rules) that generates 958 parses/sentence.

[14] Eshghi,K. Abductive Planning with Event Calculus, Proc. of ILCP'88, pp562-579, 1988.

[15] Hobbs,J.R., Stickel,M.E., Appelt,D.E. and Martin,P., Interpretation as abduction, Artificial Intelligence 63, pp69-142, 1993.

[16] Kakas,A.C., Kowalski,R.A. and Toni,F., Abductive Logic Programming, J. Logic Computation, Vol.2 No.6, pp719-770, 1992.

[17] Kakas,A.C., Kowalski,R.A. and Toni,F., The role of abduction in logic programming, Handbook of Logic in Artificial Intelligence and Logic Programming, Oxford University Press, pp235-324, 1998.

[18] Kameya,Y. and Sato,T., Efficient EM learning with tabulation for parameterized logic programs, Proc. of CL2000, LNAI 1861, Springer-Verlag, pp269-284, 2000.

[19] Koller,D. and Pfeffer,A., Learning probabilities for noisy first-order rules, Proc. of IJCAI'97, Nagoya, pp1316-1321, 1997.

[20] Koller,D., McAllester,D. and Pfeffer,A., Effective Bayesian Inference for Stochastic Programs, Proc. of AAAI'97, Rhode Island, pp740-747, 1997.

[21] Lakshmanan,L.V.S. and Sadri,F., Probabilistic Deductive Databases, Proc. of ILPS'94 pp254-268, 1994.

[22] Manning, C. D. and Schütze, H., Foundations of Statistical Natural Language Processing, The MIT Press, 1999.

[23] McLachlan, G. J. and Krishnan, T., The EM Algorithm and Extensions, Wiley Interscience, 1997.

[24] Muggleton,S., Stochastic Logic Programs, in Advances in Inductive Logic Programming (Raedt,L.De ed.) OSP Press, pp254-264, 1996.

[25] Ng,R. and Subrahmanian,V.S., Probabilistic Logic Programming, Information and Computation 101, pp150-201, 1992.

[26] Ngo,L. and Haddawy,P., Answering Queries from Context-Sensitive Probabilistic Knowledge Bases, Theoretical Computer Science 171, pp147-177, 1997.

[27] Nilsson,N.J., Probabilistic Logic, Artificial Intelligence 28, pp71-87, 1986.

[28] Pearl,J., Probabilistic Reasoning in Intelligent Systems, Morgan Kaufmann, 1988.

[29] Pfeffer,A., IBAL:A Probabilistic Programming Language, Proc. of IJCAI'01, pp733-740, 2001.

[30] Poole,D., Goebel,R. and Aleliunas,R., Theorist: a logical reasoning system for default and diagnosis, In Cercone,N., and McCalla., eds., The Knowledge Frontier, Springer, pp331-352, 1987.

[31] Poole,D., Probabilistic Horn abduction and Bayesian networks, Artificial Intelligence 64, pp81-129, 1993.

[32] Rabiner, L. and Juang, B. *Foundations of Speech Recognition*, Prentice-Hall, 1993.

[33] Sato,T., A Statistical Learning Method for Logic Programs with Distribution Semantics, Proc. of ICLP'95, pp715-729, 1995.

[34] Sato,T. and Kameya,Y., PRISM:A Language for Symbolic-Statistical Modeling, Proc. of IJCAI'97, pp1330-1335, 1997.

[35] Sato,T., Modeling Scientific Theories as PRISM Programs, ECAI Workshop on Machine Discovery, pp37-45, 1998.

[36] Sato,T., Parameterized Logic Programs where Computing Meets Learning, Proc. of FLOPS2001, LNCS 2024, 2001, pp40-60.

[37] Sato,T., Kameya,Y., Abe,S. and Shirai,K., Fast EM learning of a Family of PCFGs, Titech Technical Report (Dept. of CS) TR01-0006, Tokyo Institute of Technology, 2001.

[38] Sato,T. and Kameya, Y., Parameter Learning of Logic Programs for Symbolic-statistical Modeling, submitted for publication.

[39] Sato,T., Abe,S., Kameya,Y. and Shirai,K., A Separate-and-Learn Approach to EM Learning of PCFGs, Proc. of NLPRS2001, Tokyo, 2001 (to appear).

[40] Sakama,T. and Inoue,K., Representing Priorities in Logic Programs, Proc. of JICSLP'96, MIT Press, pp82-96, 1996.

[41] Shanahan,M., Prediction is Deduction but Explanation is Abduction, Proc. of IJCAI'89, pp1055-1060,1989.

[42] Tamaki, H. and Sato, T., OLD resolution with tabulation, Proc. of ICLP'86, LNCS 225, pp84-98, 1986.

[43] Wetherell,C.S., Probabilistic Languages: A Review and Some Open Questions, Computing Surveys, Vol.12,No.4, pp361-379, 1980.

[44] White,H.C., An Anatomy of Kinship, Prentice-Hall INC., 1963.