A Viterbi-like algorithm and EM learning for statistical abduction

Taisuke SATO **Yoshitaka KAMEYA**
Dept. of Computer Science, Graduate School of Information Science and Engineering
Tokyo Institute of Technology
Ookayama 2-12-1 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 and learning probabilistic knowledge. It combines logical abduction with a parameterized distribution over abducibles. We show that probability computation, a Viterbi-like algorithm and EM learning for statistical abduction achieve the same efficiency as specilzed algorithms for HMMs (hidden Markov models), PCFGs (probabilistic context-free grammars) and sc-BNs (singly connetcted Bayesian networks).

1 Introduction

Abduction is a form of inference that generates best hypotheses explaining an observed fact. For example, if one notices a fact 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, planning, natural language processing, etc in AI.

It is straightforward to formalize (part of) abduction in the logic programming framework [6, 9]. We have a background theory T consisting of clauses[1] and an observed fact G (usually a ground atom[2]) to be explained, and the task of abduction is to generate a hypothesis $H = \{h_1, \ldots, h_n\}$, i.e. an *explanation*, by choosing h_is from a particular set of hypothesis candidates called *abducibles*,[3] such that $T \cup H \vdash G$ and $T \cup H$ is consistent.[4] The quality of H, the abduced hypothesis, is evaluated by various criteia such as simplicity, abduction cost, and so on [9, 8, 18].

While the above framework is 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 hypothesis should be treated as being true only to some degree. Also it must be noticed that our obsevations are always finite but potentially infinite (we may have another observation indefinitely), and we would like to know the eventual outcome of infinitely many observations. It is therefore natural, even desirable, to build a new abductive framework that unifies the logical framework with a statistical framework such as Bayesian networks.

This amalgamation is made possible by introducing probabilistic elements into logic programs. More exactly, it is made possible by introducing a (parameterized) *probability distribution over abducibles*, which enables us to identify the best hypothesis as the most likely hypothesis and the likelihood can be maximized by statistical learning. The resulting logical-statistical system will benefit both abductive logic programming and Bayesian networks, as the former is expected to become robust against noise and missing data and the latter gets first-order expressiveness as knowledge representation language. We term this unification *statistical abduction.*

There are however theoretical and practical problems to achieve the unification. First of all, statistical abduction must deal with infinitely many objects sanctioned by the language of first-order logic and their joint distribution, which gives rise to a theoretical problem of constructing an infinite joint distribution of structured objects. It requires to extend the realm of traditional Bayesian networks considerably.

Secondly, to use statistical abduction in practice, we need to know all values of statistical parameters just

[1] Syntactically, a clause (in implicative form) is a logical formula $L_0 \leftarrow L_1 \wedge \ldots \wedge L_m$ ($m \geq 0$) where L_i ($0 \leq i \leq m$) is an atom or its negation, and all variables are universally quantified at the outermost position. L_0 is called a head and $L_1 \wedge \ldots \wedge L_m$ the body. A *definite* clause is one such that every L_i is an atom. Note that we adopt Prolog conventions in this paper, so a conjunction $A \wedge B$ is expressed as (A, B), and variables begin with an upper case letter.

[2] A ground atom is one that contains no variables.

[3] Usually abducibles are ground atoms.

[4] Induction also has similar formalization, and the distinction between induction and abduction is a subtle issue [7].

like conditional probabilities for Bayesian networks. Determining a large number of statistical parameters is a tough task, known as the where-do-the-numbers-come-from problem. Although one might hope that the problem is mitigated by using learning techniques, there has been little work on statistical learning in the literature of logical framework of abduction.

The objective of this paper is to point out that we have now a thereoretical foundation for statistical abduction, an efficient algorithm for computing probabilities, an efficient EM algorithm for parameter learning, and finally a Viterbi-like algorithm that generates the most likely hypothesis. Since the subject is broad and the space is limited, descriptions are necessarily sketchy, and proofs and details are left to a full paper we are preparing. Also, we assume that the reader is familiar with the basics of logic programming [6].

In what follows, after a short review of background in Sec. 2, we explain PRISM programs in Sec. 3. Sec. 4 contains the description of efficient computation of probabilities, OLDT search, EM learning and a Viterbi-like algorithm, followed by the complexity analysis of PRISM programs for HMMs, PCFGs (probabilistic context-free grammars) and sc-BNs (singly-connected Bayesian networks). Sec. 5 is a conclusion.

2 Background

The use of first-order clauses in relation to Bayesian networks has been pursued in the past decade in two strands. In the KBMC (knowledge-based model construction) approach [3, 1, 11, 14], initiated by Breese in [3], they 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). Each 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 [11]. The KBMC approach is characterized as a local approach, implicitly defining a set of local distributions as Bayesian networks, each of which corresponds to a query, evidence and context.

Statistical abduction on the other hand takes a global approach which defines a single distribution over ground atoms. It was began by Poole as "probabilistic Horn abduction" [15]. In his approach, a program contains 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 ($0 \leq i \leq n$). Abducibles in diffrent declarations are supposed to be 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 not only defines probabilities of atoms without referring to Bayesian networks but can represent Bayesian networks themselves [15].

While the probabilistic Horn abduction opened a new vista on extending Bayesian networks to first-order language, it left behind some problems. First it assumed a priori the acyclicity condition[5] and the covering property[6] of programs. These assumptions are hard to verify and could be severe restrictions in programming.[7] Also the defined probability measure is not guaranteed to be completely additive. As a result, for instance, $\lim_{n\to\infty} \mathrm{Prob}(p(t_1) \vee \ldots \vee p(t_n)) = \mathrm{Prob}(\exists X p(X))$ where the t_is are ground terms, is not guaranteed. More serious is the problem of determining parameters in disjoint declarations. How can we get them? It remained unanswered.

Aiming at providing a broader theoretical basis and a learning algorithm of uncertain parameters for the global approach, Sato proposed *distribution semantics* [19] and developed a first-order statistical modeling language PRISM[8] [19, 20]. The proposed semantics defines an *infinite joint distribution* over ground atoms as the denotation of a given program containing abducibles with statistical parameters declared just like those in Poole's disjoint declarations. It is a probabilistic extension of the least Herbrand model semantics for definite clause programs [6] to the possible world semantics with a completely additive probability measure. Since our extension is based on the least Herbrand model semantics, we don't need the covering assumption or the acyclicity condition (because every definite program has a least Herbrand model and the iff completion [6] holds in it.). Similarly, neither the range-restrictedness condition nor normalization is necessary in our approach, which is required in proof-theoretic approaches to assigning probabilities to logic programs [5, 13].

[5] 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 $A \leftarrow B_1, \ldots, B_n$.

[6] It requires that when there are finite ground instnaces $A \leftarrow \alpha_i$ ($1 \leq i \leq m$) about a ground atom A in the program, $A \Leftrightarrow \alpha_1 \vee \ldots \vee \alpha_m$ holds. The case of infinitely many ground instances is not mentioned. Intuitively the property ensures every observation has an explanation. Logically it is equivalent to assuming the iff completion [6].

[7] Under the acyclicity condition, when a clause includes local variables like Y in $p(X) \leftarrow q(X,Y), \ldots$ we cannot write recursive clauses about q such as $member(X, cons(H,Y)) \leftarrow member(X,Y)$. Also it is not clear how to check the covering property, as it specifies the relationship between what is potentially observable (which depends on nature) and what we can explain (which depends on the program).

[8] URL is `http://sato-www.cs.titech.ac.jp/prism/`

Theoretically, we may use as many constant symbols, function symbols and predicate symbols as we need, and can write whatever program. Of course, in actual programming, we have to care about efficiency, termination, etc.

Syntacticaly, our program DB is a set of clauses $F \cup R$ where F is a set of unit clauses and R is a set of definite clauses. For the purpose of theoretical explanation however, we consider DB as a set of infinitely many ground clauses made up of all possible ground instances of original clauses in DB. F then is a set of infinitely many ground atoms. We associate with F an infinite joint distribution P_F. So 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 $M(DB')$ [6]. $M(DB')$ determines the truth values of all ground atoms in DB, which implies that every ground atom A is a random variable (taking 1 when A true and 0 otherwise), and therefore a joint distribution P_{DB} for all ground atoms is definable. P_F is constructed from a collection of finite joint distritutions $P_F^{(n)}(A_1 = x_1, \ldots, A_n = x_n)$ $(n = 1, 2, \ldots)$ where A_is are random variables (abducibles). The resulting P_{DB} becomes a σ-additive probability measure over $\{0, 1\}^\omega$ (the set of all countably infinite sequences of 0 and 1).

For statistical abduction to be useful however, we need efficient methods for

- computing the probabilities of observations,
- computing the most likely explanation for the given observation, and
- learning parameters

As for HMMs, these methods correspond to the forward procedure, the Viterbi algorithm, and the Baum-Welch algorithm respectively [17].

PRISM, an implementation of the statistical abduction with P_F being constrained to a specific form, has been developed as a modeling language designed for complex symbolic statistical phenomena [19, 20]. It comes with a built-in EM algorithm[9] for parameter learning, but still lacks efficiency compared to HMMs.

In what follows, we show that there exist algorithms for PRISM programs that run as efficiently as specialized ones when they are applied to HMMs, PCFGs and sc-BNs.

3 PRISM programs

A PRISM program is defined as a definite clause program $DB = F \cup R$ which satisfies the following conditions on facts F, their distribution P_F and rules R.

[9]The EM algorithm is a standard statistical method for MLE (maximum likelihood estimation) [23].

1. F is a set of ground atoms of the form `msw`(i,n,v). The arguments i and n are called *group-id* (or *switch name*) and *trial-id*, respectively. We assume that a finite set V_i of ground terms is associated with each i, and $v \in V_i$ holds.[10] V_i corresponds to a set of values of switch i.
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 (takes the value 1) 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 each ground terms 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 $i \neq i'$.
4. Define $head(R)$ as a set of atoms appearing in the head of R. Then, $F \cap head(R) = \emptyset$.

`msw/3` in the first condition is to represent a basic probabilistic choice such as coin-tossing (`msw` stands for *multi-valued switch*). A ground atom `msw`(i,n,v) represents an event "a switch named i takes on v as a sample value on the trial n." The second and the third condition say that a logical variable `V` in `msw`$(i,n,$`V`$)$ behaves similarly to a random variable which is realized to v_k with probability θ_{i,v_k} $(k = 1 \ldots |V_i|)$.[11] 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`s are all abducibles, and the fourth condition requires that no `msw` appears in the heads of R.

As an example of representing temporal processes in an abductive framework, we describe an HMM as a PRISM program. It looks like a usual Prolog program, but denotes a discrete stochastic process (of finite length, 3 in this case).

```
1) target(hmm/1).          4) values(init,[s0,s1]).
2) data('hmm.dat').        5) values(out(_),[a,b]).
3) table([hmm/1,hmm/3]).   6) values(tr(_),[s0,s1]).

7) hmm(Cs):- msw(init,null,Si),hmm(1,Si,Cs).
8) 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).
9) hmm(T,_,[]):- T>3.
```

Procedurally, the above HMM program simulates the generation process of strings. Clauses `7`∼`9` represent the probabilistic behavior of the HMM. In clause `8`, to output a symbol `C`, we use different switches

[10]We consider DB as a countably infinite set of ground clauses, and i and n are arbitrary ground terms in the Herbrand universe.

[11]Recall however that, in distribution semantics, logical variables themselves are not random variables.

out(S) conditioned on the state S.[12] Note that T in msw(out(S),T,C) is used to guarantee independency among the choices at each time step. Recursive clauses like 8 are allowed in PRISM. Clauses 1∼6 contain additional information about the program (they are called *control declarations*). Clause 1 declares only ground atoms containing hmm/1 are observable.[13] hmm([a,b,a]) being true means this HMM generates the string aba. Clause 2 specifies a file storing learning data. Clause 3 specifies the *table predicates* (described later) are hmm/1 and hmm/3. We can read that $V_{\texttt{init}} = \{\texttt{s0},\texttt{s1}\}$, $V_{\texttt{out}(\cdot)} = \{\texttt{a},\texttt{b}\}$, $V_{\texttt{tr}(\cdot)} = \{\texttt{s0},\texttt{s1}\}$ from clauses 4∼6.

4 Three basic tasks

To apply statistical abduction to the real world, we consider, based on the analogy of HMMs [17], three basic tasks: (1) computing $P_{DB}(G=1|\vec{\theta})$,[14] the probability of an atom G representing an observation, (2) finding S^*, the most likely explanation for G, and (3) adjusting the parameters to maximize the probability of a given sequence $\mathcal{G} = \langle G_1, G_2, \ldots, G_T\rangle$ of observations. All solutions should be computationally tractable.

Poole [15] described a method for the first task. Let us consider the following if-and-only-if (iff) relation under $comp(R)$, the Clark's completion of the rules R [6]:

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

He assumes that all abducibles in $S^{(j)}$ (a finite conjunction of abducibles) are independent and $S^{(1)}, \ldots, S^{(m)}$ are exclusive to each other (we say DB satisfies the *exclusiveness condition*). Let $\psi_{DB}(G)$[15] be $\{S^{(1)}, \cdots, S^{(m)}\}$ and $\sigma_{i,v}(S)$ the number of $\texttt{msw}(i,\cdot,v)$s occurring in S. His method for solving the first task is formulated in our notation as follows:

$$P_{DB}(G=1|\vec{\theta}) = \textstyle\sum_{S\in\psi(G)} \prod_{i\in I, v\in V_i} \theta_{i,v}^{\sigma_{i,v}(S)}.$$

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

$$S^* = \arg\max_{S\in\psi(G)} \textstyle\prod_{i\in I, v\in V_i} \theta_{i,v}^{\sigma_{i,v}(S)}.$$

Unfortunately, $|\psi(G)|$, 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).

[12] Generally, a CPT of a random variable X is representable as a switch $\texttt{msw}(f_X(c_1,c_2,\ldots,c_n),\cdot,x)$, where f_X is the id of X, n is the number of conditional variables, c_i $(i = 1, \ldots, n)$ is the value of each conditional variable C_i, and x is the value of X. Of course, X's possible values should be declared in advance like Clause 4∼6.

[13] This is just for the ease of implementation.

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

[15] For simplicity, we hereafter omit the subscript DB (Similar abbreviation may occur later).

4.1 OLDT search for factorized explanations

In order to derive efficient algorithms for the three tasks, comparable to the forward procedure and the Viterbi algorithm, we need to add a couple of assumptions about programs. We assume that DB satisfies the exclusiveness condition, and that m is finite in Eq. 1 (we say DB satisfies the *finite support condition*).[16] Furthermore, we assume that the following iff-relational formula holds for some finite ordered set $\tau_{DB}(G) = \{\tau_1, \ldots, \tau_K\}$ of table atoms (atoms containing the table predicate[17]):

$$\begin{aligned} comp(R) \;\models\; & (G \leftrightarrow S_{0,1} \vee \cdots \vee S_{0,m_0}) \\ & \wedge (\tau_1 \leftrightarrow S_{1,1} \vee \cdots \vee S_{1,m_1}) \\ & \wedge \cdots \wedge (\tau_{K_t} \leftrightarrow S_{K,1} \vee \cdots \vee S_{K,m_K}), \end{aligned} \qquad (2)$$

where, letting G be a special table atom τ_0, each in $\tilde{\psi}_{DB}(\tau_k) \stackrel{\text{def}}{=} \{S_{k,1}, \ldots, S_{k,m_k}\}$ $(1 \le k \le K)$ is a subset of $F \cup \{\tau_{k+1}, \ldots, \tau_K\}$ called a *factorized-*(or *tabled-*)*explanation* for τ_k, and each of $S_{k,1}, \ldots, S_{k,m_k}$ $(0 \le k \le K)$ is a set of independent atoms.

$\tilde{\psi}_{DB}(\tau_k)$ and $\tau_{DB}(G)$ can be obtained by OLDT search[18][22], a complete refutation technique with tabulation for logic programs. We first translate the source PRISM program to Prolog similarly to *definite clause grammars* (DCGs) [21], or Poole's *Theorist* [16]. The following Prolog program is a translation of the HMM program in Sec. 3. Clauses Tj and Tj' are generated from the clause j:

```
T1)  top_hmm(Cs,X):- tab_hmm(Cs,X,[]).
T3)  tab_hmm(Cs,[hmm(Cs)|X],X):- hmm(Cs,_,[]).
T3') tab_hmm(T,S,Cs,[hmm(T,S,Cs)|X],X):-
         hmm(T,S,Cs,_,[]).
T4)  e_msw(init,T,s0,[msw(init,T,s0)|X],X).
T4') e_msw(init,T,s1,[msw(init,T,s1)|X],X).
  :
T7)  hmm(Cs,X0,X1):-
         e_msw(init,null,Si,X0,X2),
         tab_hmm(1,Si,Cs,X2,X1).
T8)  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).
T9)  hmm(T,S,[],X,X):- T>3.
```

As can be seen, in translation, we add two arguments to collect factorized explanations and rename the predicates of abducibles (msws) and table atoms. We then perform OLDT search, noting extra arguments do not influence the search procedure, while saving in the solution table a list of factorized explanations every time

[16] This condition says that every observation has a finite number of explanations. Without it, our search for explanations would not terminate.

[17] Table predicates are assumed to be declared by the programmer in advance, like Clause 3 in the HMM program.

[18] OLDT stands for *Ordered Linear resolution for Definite clauses with Tabulation.*

```
hmm([a,b,a]): [hmm([a,b,a]):[[msw(init,null,s0),hmm(1,s0,[a,b,a])],
                             [msw(init,null,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 1: Solution table for the observation `hmm([a,b,a])`.

the table atom is solved. For example, the OLDT search for the above translation yields Fig. 1. Finally, we extract $\tilde{\psi}$, the set of all factorized explanations from the solution table. The remaining task is to get totally ordered table atoms, i.e. the ordered set $\tau_{DB}(G)$, respecting the acyclicity in Eq. 2, which can be done by *topological sorting.*

4.2 Computing the observation probability and the most likely explanation

Given the iff-relational formula in Eq. 2, an efficient algorithm for computing $P_{DB}(G=1|\vec{\theta})$ (the first task) is derivable based on the analogy of the *inside probabilities* in Baker's Inside-Outside algorithm [2]. In our formulation however, the inside probability of a table atom τ is $P_{DB}(\tau=1|\vec{\theta})$, the probability of τ being true. It should be noted that, when computing $\mathcal{P}[\tau_k]$, the inside probabilities $\mathcal{P}[\tau_K], \mathcal{P}[\tau_{K-1}], \ldots, \mathcal{P}[\tau_{k+1}]$ have been already computed. The computation terminates with $\mathcal{P}[G](=\mathcal{P}[\tau_0])$, the inside probability of G.

procedure GET-INSIDE-PROBS (DB, G) **begin**
  Put $G = \tau_0$;
  for $k := K$ **downto** 0 **do**
    $P[\tau_k] :=$
      $\sum_{S\in\tilde{\psi}(\tau_k)} \prod_{\mathtt{msw}(i,\cdot,v)\in S} \theta_{i,v} \prod_{\tau\in S\cap\{\tau_{k+1},\ldots,\tau_K\}} \mathcal{P}[\tau]$
end.

Similarly, an efficient algorithm for computing the most likely explanation S^* (i.e. the second task) is derived. The algorithm GET-ML-EXPL first computes $\delta[\tau_k]$, the maximum probability of factorized explanations for each table atom τ_k. $\mathcal{E}[\tau_k]$, the most likely factorized explanation for τ_k, is then constructed. Finally, we construct S^* from $\mathcal{E}[\cdot]$. For an HMM program (see Sec. 3), GET-ML-EXPL is equivalent to the Viterbi algorithm, a standard algorithm for finding the most likely state-transition path of HMMs. Furthermore, when we write a probabilistic context-free grammar (PCFG) in PRISM, it is easily shown that probabilistic parsers for PCFGs, which finds most likely parse of a given sentence, are equivalent to GET-ML-EXPL.

procedure GET-ML-EXPL(DB, G) **begin**
  Put $G = \tau_0$;
  for $k := K$ **downto** 0 **do begin**
    foreach $S \in \tilde{\psi}(\tau_k)$ **do**
      $\delta'[\tau_k, S] :=$
        $\prod_{\mathtt{msw}(i,\cdot,v)\in S} \theta_{i,v} \prod_{\tau\in S\cap\{\tau_{k+1},\ldots,\tau_K\}} \delta[\tau]$;
    $\delta[\tau_k] := \max_{S\in\tilde{\psi}(\tau_k)} \delta'[\tau_k, S]$;
    $\mathcal{E}[\tau_k] := \arg\max_{S\in\tilde{\psi}(\tau_k)} \delta'[\tau_k, S]$
  end;
  $s := \{G\}$; $S^* := \emptyset$;
  while $s \neq \emptyset$ **do begin**
    Select and remove A from s;
    if $A = \mathtt{msw}(\cdot,\cdot,\cdot)$ **then** add A to S^*
    else $s := s \cup \mathcal{E}[A]$
  end
end

4.3 Learning algorithm

The learning of PRISM programs means MLE (maximum likelihood estimation) of statistical parameters of `msw`s embedded in a program, for which the EM algorithm is appropriate [23]. A new EM algorithm named *graphical EM algorithm* that takes advantage of the structure of the iff-relational formula in Eq. 2 has been derived on the analogy of computation of the *outside probabilities* in the Inside-Outside algorithm [10]. We added an assumption that all observable atoms are exclusive to each other and the probabilities of these atoms sum up to one (we say DB satisfies the *uniqueness condition*).

The graphical EM algorithm first initializes the parameters $\vec{\theta}$ and then iterates re-estimation until the likelihood saturates. In each iteration, the *expected occurrence* is computed first for each abducible atom $\mathtt{msw}(i,n,v)$ from its parameter $\theta_{i,v}$, the inside probabilities and the outside probabilities, and then all parameters are simultaneously re-estimated by using such expected occurrences. Final values of parameters become learned ones.

4.4 Complexity

In this section, we primarily evaluate time complexity of our methods for the first and the second task due to space limitations. The method for the first (resp. the second) task comprises two phases — OLDT search and GET-INSIDE-PROBS (resp. GET-ML-EXPL). Hence, we should estimate each algorithm separately. Assuming that all table operations can be performed in $O(1)$ time, the computation time of OLDT search is measured by the size of the search tree. As for the computation time of GET-INSIDE-PROBS and GET-ML-EXPL, it is $O(\xi_{\text{num}}\xi_{\text{maxsize}})$,[19] where $\xi_{\text{num}} \stackrel{\text{def}}{=} |\Delta|$, $\xi_{\text{maxsize}} \stackrel{\text{def}}{=} \max_{S\in\Delta}|S|$, and $\Delta \stackrel{\text{def}}{=} \bigcup_{\tau\in\tau_{DB}} \tilde{\psi}_{DB}(\tau)$. In the case of the HMM program, $\xi_{\text{num}} = O(N^2L)$ and $\xi_{\text{maxsize}} = O(1)$. Hence, the computation time of GET-INSIDE-PROBS and GET-ML-EXPL is $O(N^2L)$. This is the same order as that of the forward procedure and the Viterbi algorithm. So GET-ML-EXPL (resp. GET-INSIDE-PROBS) is a generalization of the Viterbi algorithm (resp. the forward procedure). For existing symbolic-statistical models, we summarize computation time as follows:[20]

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

The second (resp. the third) column"OLDT time" (resp. "GIP/GMLE time") indicates that the computation time of OLDT search (resp. GET-INSIDE-PROBS and GET-ML-EXPL) for the model in the first column. N, L, T, $|V|$ and M are the number of states of the target HMM, the maximum length of input strings, the size of training data, the number of nodes in the target Bayesian network, and the number of non-terminal symbols in the target PCFG, respectively. The above table exemplifies that our general framework can subsume specific algorithms such as the Viterbi algorithm.

5 Conclusion

We have proposed statistical abduction as the combination of abductive logic programming with a distribution over abducibles. It has first-order expressive power and integrates current most powerful probabilistic knowledge representation frameworks such as HMMs, PCFGs and Bayesian networks. Besides, our general algorithms developed for the three tasks (probability computation, the search for the most likely explanation, and EM learning) achieve the same efficiency as specialized algorithms for above three frameworks.

Acknowledgments

This research is supported by Grant-in-Aid for Scientific Research on Priority Area "Discovery Science" 1999 of the Ministry of Education, Science, Sports and Culture, Japan.

[19] By the way, our method for the third task (EM learning) comprises OLDT search and the graphical EM algorithm whose computation is measured by the re-estimation time (since we do not know the number of re-estimations in advance). It is shown, analogously to GET-INSIDE-PROBS and GET-ML-EXPL, to be $O(\xi_{\text{num}}\xi_{\text{maxsize}}T)$.

[20] In PCFGs, we assume grammars are in Chomsky normal form, and in sc-BNs, we assume the maximum number of parent nodes are fixed.

References

[1] Bacchus, F. Using first-order probability logic for the construction of Bayesian networks. *Proc. of UAI'93*, 1993.

[2] Baker, J. K. Trainable grammars for speech recognition. *Proc. of Spring Conference of the Acoustical Society of America*, 1979.

[3] Breese, J. S. Construction of belief and decision networks. *Comput. Intell.*, 8(4), 1992.

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

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

[6] Doets, K. *From Logic to Logic Programming*. The MIT Press, Cambridge, 1994.

[7] Flach, P. and Kakas, A. (eds). *Abduction and Induction – essays on their relation and integration –*. Kluwer Academic Publishers, 2000.

[8] Hobbs, J. R., Stickel, M. E., Appelt, D. E. and Martin, P. Interpretation as abduction. *Artif. Intell.*, 63, 1993.

[9] Kakas, A. C., Kowalski, R. A. and Toni, F. Abductive logic programming. *J. Logic Comput.*, 2(6), 1992.

[10] Kameya, Y. and Sato, T., Efficient EM learning with tabulation for parameterized logic programs, to be presented at the 1st Conf. on Computational Logic (CL2000), 2000.

[11] Koller, D. and Pfeffer, A. Learning probabilities for noisy first-order rules. *Proc. of IJCAI'97*, 1997.

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

[13] Muggleton, S. Stochastic logic programs. In *Advances in Inductive Logic Programming* (Raedt, L. De ed.), OSP Press, 1996.

[14] Ngo, L. and Haddawy, P. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171, 1997.

[15] Poole, D. Probabilistic Horn abduction and Bayesian networks. *Artif. Intell.*, 64, 1993.

[16] Poole, D., Compiling a default reasoning system into Prolog, *New Generation Comput.*, 9(1), 1991.

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

[18] Sakama,T. and Inoue,K., Representing Priorities in Logic Programs, *Proc. of JICSLP96*, MIT Press, 1996.

[19] Sato, T. A statistical learning method for logic programs with distribution semantics. *Proc. of ICLP'95*, 1995.

[20] Sato, T. and Kameya, Y. PRISM: a language for symbolic-statistical modeling. *Proc. of IJCAI'97*, 1997.

[21] Sterling, L. and Shaprio, E. *The Art of Prolog*, The MIT Press, 1986.

[22] Tamaki, H. and Sato, T. OLD resolution with tabulation. *Proc. of ICLP'86*, LNCS 225, 1986.

[23] Tanner, M. *Tools for Statistical Inference* (2nd ed.). Springer-Verlag, 1986.