Errata to Foundations of Probabilistic Logic Programming

Fabrizio Riguzzi

Page 154

The text:

**Binary Decision Diagrams (BDDs)** perform a Shannon expansion of the Boolean formula: they express the formula as

\[ f_K(X) = X_1 \lor f_K^{X_1}(X) \land \neg X_1 \lor f_K^{X_1}(X) \]

should be replaced by

**BDDs** perform a Shannon expansion of the Boolean formula: they express the formula as

\[ f_K(X) = X_1 \land f_K^{X_1}(X) \lor \neg X_1 \land f_K^{X_1}(X) \]

Page 161

The text:

The Boolean variables are associated with the following parameters:

\[
\begin{align*}
P(X_{ij1}) &= P(X_{ij1} = 1) \\
& \quad \ldots \\
P(X_{ijk}) &= \frac{P(X_{ij} = k)}{\prod_{l=1}^{k-1} (1 - P(X_{ijkl-1}))}
\end{align*}
\]

should be replaced by

The Boolean variables are associated with the following parameters:

\[
\begin{align*}
P(X_{ij1}) &= P(X_{ij1} = 1) \\
& \quad \ldots \\
P(X_{ijk}) &= \frac{P(X_{ij} = k)}{\prod_{l=1}^{k-1} (1 - P(X_{ijl}))}
\end{align*}
\]
Page 174

The text:

To define structured decomposability, consider a Deterministic Decomposable Negation Normal Form (d-DNNF) $\delta$ and assume, without loss of generality, that all conjunctions are binary. $\delta$ respects a vtree $V$ if for every conjunction $\alpha \land \beta$ in $\delta$, there is a node $v$ in $V$ such that $\text{vars}(\alpha) \subseteq \text{vars}(v_l)$ and $\text{vars}(\beta) \subseteq \text{vars}(v_r)$ where $v_l$ and $v_r$ are the left and right child of $v$. $\delta$ enjoys structured decomposability if it satisfies some vtree.

Page 176, Definition 35

The text:

Definition 35 ($Tc_P$ operator [Vlasselaer et al., 2015, 2016]). Let $P$ be a ground probabilistic logic program with probabilistic facts $F$ and atoms $B_P$. Let $I$ be a parameterized interpretation with pairs $(a, \lambda_a)$. Then, the $Tc_P$ operator is $Tc_P(I) = \{(a, \lambda_a) \mid a \in B_P\}$ where

$$\lambda'_a = \begin{cases} a & \text{if } a \in F \\ \bigvee_{a \in \text{B}_P \setminus F} (\lambda_{b_1} \land \ldots \land \lambda_{b_n} \land \neg \lambda_{c_1} \land \ldots \land \neg \lambda_{c_m}) & \text{if } a \in B_P \setminus F \end{cases}$$

should be replaced by

Definition 35 ($Tc_P$ operator [Vlasselaer et al., 2015, 2016]). Let $P$ be a ground probabilistic logic program with probabilistic facts $F$, rules $R$ and atoms $B_P$. Let $I$ be a parameterized interpretation with pairs $(a, \lambda_a)$. Then, the $Tc_P$ operator is $Tc_P(I) = \{(a, \lambda_a) \mid a \in B_P\}$ where

$$\lambda'_a = \begin{cases} a & \text{if } a \in F \\ \bigvee_{a \in \text{B}_P \setminus F} (\lambda_{b_1} \land \ldots \land \lambda_{b_n} \land \neg \lambda_{c_1} \land \ldots \land \neg \lambda_{c_m}) & \text{if } a \in B_P \setminus F \end{cases}$$
Page 177

The text:

\textit{Vlasselaer et al. [2016]} show that if each atom is selected frequently enough in step 1, then the same fixpoint $\text{lfp}(T_{CP})$ is reached as for the naive algorithm, provided that the operator is still applied stratum by stratum in normal logic programs.

should be replaced by

\textit{Vlasselaer et al. [2016]} show that if each atom is selected frequently enough in step 1, then the same fixpoint $\text{lfp}(T_{CP})$ is reached as for the naive algorithm that considers all atoms at the same time, provided that the operator is still applied stratum by stratum in normal logic programs.

Page 247, Algorithm 11

The text:
Algorithm 11 Function EXACTSOLUTION: Solving the DTProBLOG decision problem exactly.

1: function EXACTSOLUTION(\( DT \))
2: \( \text{ADD}_{\text{util}} \leftarrow 0 \)
3: for all \( (u \rightarrow r) \in l/ \) do
4: Build BDD(\( u \)), the BDD for \( u \)
5: \( \text{ADD}(u) \leftarrow \text{PROBABILITYDD}(\text{BDD}_u(\text{DT})) \)
6: \( \text{ADD}_{\text{util}}^u(u) \leftarrow r \cdot \text{ADD}(u) \)
7: \( \text{ADD}_{\text{tot}}^u \leftarrow \text{ADD}_{\text{util}}^u \oplus \text{ADD}_{\text{util}}(u) \)
8: end for
9: let \( t_{\text{max}} \) be the terminal node of \( \text{ADD}_{\text{tot}}^u \) with the highest utility
10: let \( p \) be a path from \( t_{\text{max}} \) to the root of \( \text{ADD}_{\text{tot}}^u \)
11: return the Boolean decisions made on \( p \)
12: end function
13: function PROBABILITYDD(\( n \))
14: if \( n \) is the 1-terminal then
15: return a 1-terminal
16: end if
17: if \( n \) is the 0-terminal then
18: return a 0-terminal
19: end if
20: let \( h \) and \( l \) be the high and low children of \( n \)
21: \( \text{ADD}_h \leftarrow \text{PROBABILITYDD}(h) \)
22: \( \text{ADD}_l \leftarrow \text{PROBABILITYDD}(h) \)
23: if \( n \) represents a decision \( d \) then
24: return ITE\( (d, \text{ADD}_h, \text{ADD}_l) \)
25: end if
26: if \( n \) represents a fact with probability \( p \) then
27: return \( (p \cdot \text{ADD}_h) \oplus ((1 - p) \cdot \text{ADD}_l) \)
28: end if
29: end function

should be replaced by
Algorithm 11 Function `EXACTSOLUTION`: Solving the DTProBLOG decision problem exactly.

1: function `EXACTSOLUTION(DT)`
2: \[ \text{ADD}_{\text{tot}}^{\text{util}} \leftarrow 0 \]
3: for all \((u \rightarrow r) \in \mathcal{I}/ \mathcal{D}\) do
4: \[ \text{Build BDD}(u), \text{the BDD for } u \]
5: \[ \text{ADD}(u) \leftarrow \text{PROBABILITYDD}(\text{BDD}(u)) \]
6: \[ \text{ADD}_{\text{tot}}^{\text{util}}(u) \leftarrow r \cdot \text{ADD}(u) \]
7: \[ \text{ADD}_{\text{tot}}^{\text{util}} \leftarrow \text{ADD}_{\text{tot}}^{\text{util}} \oplus \text{ADD}_{\text{tot}}^{\text{util}}(u) \]
8: end for
9: let \(t_{\text{max}}\) be the terminal node of \(\text{ADD}_{\text{tot}}^{\text{util}}\) with the highest utility
10: let \(p\) be a path from \(t_{\text{max}}\) to the root of \(\text{ADD}_{\text{tot}}^{\text{util}}\)
11: return the Boolean decisions made on \(p\)
12: end function

13: function `PROBABILITYDD(n)`
14: if \(n\) is the 1-terminal then
15: \[ \text{return a 1-terminal} \]
16: end if
17: if \(n\) is the 0-terminal then
18: \[ \text{return a 0-terminal} \]
19: end if
20: let \(h\) and \(l\) be the high and low children of \(n\)
21: \[ \text{ADD}_h \leftarrow \text{PROBABILITYDD}(h) \]
22: \[ \text{ADD}_l \leftarrow \text{PROBABILITYDD}(l) \]
23: if \(n\) represents a decision \(d\) then
24: \[ \text{return ITE}(d, \text{ADD}_h, \text{ADD}_l) \]
25: end if
26: if \(n\) represents a fact with probability \(p\) then
27: \[ \text{return } (p \cdot \text{ADD}_h) \oplus ((1 - p) \cdot \text{ADD}_l) \]
28: end if
29: end function

Pages 260-261

The text:

To perform Expectation Maximization (EM), we can associate a random variable \(X_{ij}\) with values \(D = \{x_{i1}, \ldots, x_{in_i}\}\) to the ground switch name \(i\theta_j\) of \(msw(i, x)\) with domain \(D\), with \(\theta_j\) being a grounding substitution for \(i\). Let \(g(i)\) be the set of such substitutions:

\[ g(i) = \{j | \theta_j \text{ is a grounding substitution for } i \text{ in } msw(i, x) \} \]

The EM algorithm alternates between the two phases:

- Expectation: computes \(E[c_{ik} | e]\) for all examples \(e\), switches \(msw(i, x)\) and \(k \in \{1, \ldots, n_i\}\), where \(c_{ik}\) is the number of times a variable \(X_{ij}\)
takes value $x_{ik}$ with $j$ in $g(i)$. \( E[c_{ik}|e] \) is given by \( \sum_{j \in g(i)} P(X_{ij} = x) \).

- **Maximization**: computes \( \Pi_{ik} \) for all \( msx(i, x) \) and \( k = 1, \ldots, n_i - 1 \) as

\[
\Pi_{ik} = \frac{\sum_{e \in E} E[c_{ik}|e]}{\sum_{k=1}^{n_i} E[c_{ik}|e]}
\]

So, for each example \( e, X_{ij}s \) and \( x_{iks} \), we compute \( P(X_{ij} = x_{ik}|e) \), the expected value of \( X_{ij} \) given the example, with \( k \in \{1, \ldots, n_i\} \). These expected values are then aggregated and used to complete the dataset for computing the parameters by relative frequency. If \( c_{ijk} \) is number of times a variable \( X_{ij} \) takes value \( x_{ik} \) for any \( j \), \( E[c_{ik}|e] \) is its expected value given example \( e \). If \( E[c_{ik}] \) is its expected value given all the examples, then

\[
E[c_{ik}] = \sum_{t=1}^{T} E[c_{ik}|e_t]
\]

and

\[
\Pi_{ik} = \frac{E[c_{ik}]}{\sum_{k=1}^{n_i} E[c_{ik}]}
\]

should be replaced by

To perform EM, we can associate a random variable \( X_{ij} \) with values \( D = \{x_{i1}, \ldots, x_{in_i}\} \) to the ground switch name \( i\theta_j \) of \( msx(i, x) \) with domain \( D \), with \( \theta_j \) being a grounding substitution for \( i \). Let \( g(i) \) be the set of such substitutions:

\[
g(i) = \{j|\theta_j \text{ is a grounding substitution for } i \text{ in } msx(i, x)\}.
\]

PRISM will learn different parameters for each \( X_{ij} \) random variable. The EM algorithm alternates between the two phases:

- **Expectation**: computes \( E[c_{ijk}|e] \) for all examples \( e \), switches \( msx(i \theta_j, x) \) and \( k \in \{1, \ldots, n_i\} \), where \( c_{ijk} \) is the number of times variable \( X_{ij} \) takes value \( x_{ik} \). \( E[c_{ijk}|e] \) is given by \( P(X_{ij} = x_{ik}|e) \).

- **Maximization**: computes \( \Pi_{ijk} \) for all \( msx(i \theta_j, x) \) and \( k = 1, \ldots, n_i - 1 \) as

\[
\Pi_{ijk} = \frac{\sum_{e \in E} E[c_{ijk}|e]}{\sum_{k=1}^{n_i} E[c_{ijk}|e]}
\]

So, for each example \( e, X_{ij}s \) and \( x_{iks} \), we compute \( P(X_{ij} = x_{ik}|e) \), the expected value of \( X_{ij} \) given the example, with \( k \in \{1, \ldots, n_i\} \). These expected values are then used to complete the dataset for computing the parameters by relative frequency. If \( c_{ijk} \) is number of times a variable \( X_{ij} \) takes value \( x_{ik} \), \( E[c_{ijk}|e] \) is its expected value given example \( e \). If \( E[c_{ijk}] \) is its expected value given all the examples, then

\[
E[c_{ijk}] = \sum_{t=1}^{T} E[c_{ijk}|e_t]
\]
and
\[ \Pi_{ijk} = \frac{E[c_{ijk}]}{\sum_{k=1}^{m} E[c_{ijk}]} . \]

Page 262-263, Algorithms 13-14

The text:

Algorithm 13 Function PRISM-EM: Naive EM learning in PRISM

1: function PRISM-EM-NAIVE(\(E\), \(P\), \(\epsilon\))
2: \(LL = -\inf\)
3: repeat
4: \(LL_0 = LL\)
5: for all \(i, k\) do \(\triangleright \text{Expectation step}\)
6: \(E[c_{ik}] \leftarrow \sum_{\nu \in E} \sum_{\omega \in \text{msw}(i, v_k) \delta_{\omega} \nu} P(\kappa) P(\omega)\)
7: end for
8: for all \(i, k\) do \(\triangleright \text{Maximization step}\)
9: \(\Pi_{ik} \leftarrow \frac{E[c_{ik}]}{\sum_{k=1}^{m} E[c_{ik}]}\)
10: end for
11: \(LL \leftarrow \sum_{\omega \in E} \log P(\omega)\)
12: until \(LL - LL_0 < \epsilon\)
13: return \(LL, \Pi_{ik}\) for all \(i, k\)
14: end function

Algorithm 14 Procedure GET-INSIDE-PROBS: computation of inside probabilities.

1: procedure GET-INSIDE-PROBS(q)
2: for all \(i, k\) do
3: \(P(\text{msw}(i, v_k)) \leftarrow \Pi_{ik}\)
4: end for
5: for \(i \leftarrow m \rightarrow 1\) do
6: \(P(g_i) \leftarrow 0\)
7: for \(j \leftarrow 1 \rightarrow s_i\) do
8: Let \(S_{ij}\) be \(h_{ij}, \ldots, h_{ijo}\)
9: \(P(g_i, S_{ij}) \leftarrow \prod_{l=1}^{s} P(h_{ijl})\)
10: \(P(g_i) \leftarrow P(g_i) + P(g_i, S_{ij})\)
11: end for
12: end for
13: end procedure

should be replaced by
Algorithm 13 Function PRISM-EM-NAIVE: Naive EM learning in PRISM

1: function PRISM-EM-NAIVE($E$, $P$, $\epsilon$)  
2: $LL = -\inf$  
3: repeat  
4: $LL_0 = LL$  
5: for all $i,j,k$ do  
6: $E[ijk] = \sum_{e \in E} \sum_{\kappa \in K_{\text{meas}}(i\theta_j,x_k)e} \epsilon \cdot P(e)$  
7: end for  
8: for all $i,j,k$ do  
9: $\pi[ijk] = \frac{\sum_{e \in E} E[ijk]}{\sum_{\kappa \in K_{\text{meas}}(i\theta_j,x_k)e} \epsilon \cdot P(e)}$  
10: end for  
11: $LL = \sum_{e \in E} \log P(e)$  
12: until $LL - LL_0 < \epsilon$  
13: return $LL$, $\pi[ijk]$ for all $i,j,k$  
14: end function

Algorithm 14 Procedure GET-INSIDE-PROBS: computation of inside probabilities.

1: procedure GET-INSIDE-PROBS($q$)  
2: for all $i,j,k$ do  
3: $P(\text{msw}(i\theta_j,v_k)) \leftarrow \pi[ijk]$  
4: end for  
5: for $i \leftarrow m \rightarrow 1$ do  
6: $P(g_i) \leftarrow 0$  
7: for $j \leftarrow 1 \rightarrow s_i$ do  
8: Let $S_{ij}$ be $h_{ij1}, \ldots, h_{ijo}$  
9: $P(g_i, S_{ij}) \leftarrow \prod_{l=1}^{s_i} P(h_{ijl})$  
10: $P(g_i) \leftarrow P(g_i) + P(g_i, S_{ij})$  
11: end for  
12: end for  
13: end procedure

Pages 263-264

The text:

If $g_i = \text{msw}(i,x_k)\theta_j$, then

$$P(X_{ij} = x_{ik}, e) = Q(g_i)P(g_i) = Q(g_i)\pi_{ik}.$$  

In fact, we can divide the explanations for $e$ into two sets, $K_{e1}$, that includes the explanations containing $\text{msw}(i,x_k)\theta_j$, and $K_{e2}$, that includes the other explanations. Then $P(e) = P(K_{e1}) + P(K_{e2})$ and $P(X_{ij} = x_{ik}, e) = P(K_{e1})$. Since each explanation in $K_{e1}$ contains $g_i = \text{msw}(i,x_k)\theta_j$, $K_{e1}$ takes the form $\{\{g_i,W_1\}, \ldots, \{g_i,W_s\}\}$ and
should be replaced by

If \( g_i = msw(i\theta_j, x_k) \), then

\[
P(X_{ij} = x_{ik}, e) = Q(g_i) P(g_i) = Q(g_i) \Pi_{ijk}.
\]

In fact, we can divide the explanations for \( e \) into two sets, \( K_{e1} \), that includes the explanations containing \( msw(i\theta_j, x_k) \), and \( K_{e2} \), that includes the other explanations. Then \( P(e) = P(K_{e1}) + P(K_{e2}) \) and \( P(X_{ij} = x_{ik}, e) = P(K_{e1}) \). Since each explanation in \( K_{e1} \) contains \( g_i = msw(i\theta_j, x_k) \), \( K_{e1} \) takes the form \( \{\{g_i, W_1\}, \ldots, \{g_i, W_s\}\} \) and

Page 265, algorithms 16-17

The text:

Algorithm 16 Function PRISM-EM

1: function PRISM-EM(E, P, \( \epsilon \))
2: \( LL = -\infty \)
3: repeat
4: \( LL_0 = LL \)
5: \( LL = \text{EXPECTATION}(E) \)
6: for all \( i \) do
7: \( \text{Sum} \leftarrow \sum_{k=1}^{ni} E[e_{ik}] \)
8: for \( k = 1 \) to \( ni \) do
9: \( \Pi_{ik} = \frac{E[e_{ik}]}{\text{Sum}} \)
10: end for
11: end for
12: until \( LL - LL_0 < \epsilon \)
13: return \( LL, \Pi_{ik} \) for all \( i, k \)
14: end function
Algorithm 17 Procedure PRISM-EXPECTATION

1: function PRISM-EXPECTATION($E$) 
2: \[ LL = 0 \] 
3: for all $e \in E$ do 
4: \[ \text{GET-INSIDE-PROBS}(e) \] 
5: \[ \text{GET-OUTSIDE-PROBS}(e) \] 
6: for all $i$ do 
7: \[ \text{for } k = 1 \text{ to } n_i \text{ do} \] 
8: \[ E[c_{ik}] = E[c_{ik}] + Q(msw(i, x_k))\Pi_{ik}/P(e) \] 
9: \[ \text{end for} \] 
10: \[ \text{end for} \] 
11: \[ LL = LL + \log P(e) \] 
12: \[ \text{end function} \]

Algorithm 16 Function PRISM-EM

1: function PRISM-EM($E, P, \epsilon$) 
2: \[ LL = -\infty \] 
3: repeat 
4: \[ LL_0 = LL \] 
5: \[ LL = \text{EXPECTATION}(E) \] 
6: for all $i, j$ do 
7: \[ \text{Sum} \leftarrow \sum_{k=1}^{n_i} E[c_{ijk}] \] 
8: \[ \text{for } k = 1 \text{ to } n_i \text{ do} \] 
9: \[ \Pi_{ijk} = \frac{E[c_{ijk}]}{\text{Sum}} \] 
10: \[ \text{end for} \] 
11: \[ \text{end for} \] 
12: until $LL - LL_0 < \epsilon$ 
13: return $LL, \Pi_{ijk}$ for all $i, j, k$ 
14: end function

should be replaced by
Algorithm 17 Procedure PRISM-EXPECTATION

1: function PRISM-EXPECTATION($E$)  
2:   $LL = 0$  
3:   for all $e \in E$ do  
4:     GET-INSIDE-PROBS($e$)  
5:     GET-OUTSIDE-PROBS($e$)  
6:       for all $i, j$ do  
7:         for $k = 1$ to $n_i$ do  
8:           $E[c_{ijk}] = E[c_{ijk}] + Q(msw(i\theta_j, x_k))\Pi_{ijk}/P(e)$  
9:         end for  
10:       end for  
11:     $LL = LL + \log P(e)$  
12:   end for  
13: return $LL$  
14: end function

1 Page 272

The text:

$$
\pi_{ik} = \frac{\sum_{e \in E} E[c_{ik1}|e]}{\sum_{e \in E} E[c_{ik0}|e] + E[c_{ik1}|e]}
$$

should be replaced by

$$
\pi_{ik} = \frac{\sum_{e \in E} E[c_{ik1}|e]}{\sum_{e \in E} E[c_{ik0}|e] + E[c_{ik1}|e]}
$$

2 Page 281

The text:

LFI-ProbLog computes $P(X_{ij} = x|I)$ by computing $P(X_{ij} = x, I)$ using Procedure CIRCP shown in Algorithm 5: the $\mathbf{d}$-DNNF circuit is visited twice, once bottom up to compute $P(q(I))$ and once top down to compute $P(X_{ij} = x|I)$ for all the variables $X_{ij}$ and values $x$. Then $P(X_{ij} = x|I)$ is given by $\frac{P(X_{ij} = x, I)}{P(I)}$.

should be replaced by

LFI-ProbLog computes $P(X_{ij} = x|I)$ by computing $P(X_{ij} = x, I)$ using Procedure CIRCP shown in Algorithm 5: the $\mathbf{d}$-DNNF circuit is visited twice, once bottom up to compute $P(q(I))$ and once top down to compute $P(X_{ij} = x, I)$ for all the variables $X_{ij}$ and values $x$. Then $P(X_{ij} = x|I)$ is given by $\frac{P(X_{ij} = x, I)}{P(I)}$. 

11
References
