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
>
> $$f_K(\mathbf{X}) = \mathrm{X}_1 \vee f_K^{\mathrm{X}_1}(\mathbf{X}) \wedge \neg\mathrm{X}_1 \vee f_K^{\neg\mathrm{X}_1}(\mathbf{X})$$

should be replaced by

> BDDs perform a Shannon expansion of the Boolean formula: they express the formula as
>
> $$f_K(\mathbf{X}) = \mathrm{X}_1 \wedge f_K^{\mathrm{X}_1}(\mathbf{X}) \vee \neg\mathrm{X}_1 \wedge f_K^{\neg\mathrm{X}_1}(\mathbf{X})$$

Page 161

The text:

> The Boolean variables are associated with the following parameters:
>
> $$\begin{array}{rcl} P(\mathrm{X}_{ij1}) & = & P(\mathrm{X}_{ij1} = 1) \\ & \dots & \\ P(\mathrm{X}_{ijk}) & = & \dfrac{P(\mathrm{X}_{ij} = k)}{\prod_{l=1}^{k-1}(1 - P(\mathrm{X}_{ijk-1}))} \end{array}$$

should be replaced by

> The Boolean variables are associated with the following parameters:
>
> $$\begin{array}{rcl} P(\mathrm{X}_{ij1}) & = & P(\mathrm{X}_{ij1} = 1) \\ & \dots & \\ P(\mathrm{X}_{ijk}) & = & \dfrac{P(\mathrm{X}_{ij} = k)}{\prod_{l=1}^{k-1}(1 - P(\mathrm{X}_{ijl}))} \end{array}$$

Page 174

The text:

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

should be replaced by

> To define structured decomposability, consider a d-DNNF δ and assume, without loss of generality, that all conjunctions are binary. δ *respects* a vtree V if for every conjunction $\alpha \wedge \beta$ in δ, there is a node v in V such that $vars(\alpha) \subseteq vars(v_l)$ and $vars(\beta) \subseteq vars(v_r)$ where v_l and v_r are the left and right child of v and $vars(v)$ is the set of variables appearing in d-DNNF v. δ 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 $\mathcal{P}$ be a ground probabilistic logic program with probabilistic facts $\mathcal{F}$ and atoms $\mathcal{B}_\mathcal{P}$. Let $\mathcal{I}$ be a parameterized interpretation with pairs (a, λ_a). Then, the Tc_P operator is $Tc_P(\mathcal{I}) = \{(a, \lambda_a) | a \in \mathcal{B}_\mathcal{P}\}$ where*
>
> $$\lambda'_a = \begin{cases} a & \text{if } a \in \mathcal{F} \\ \bigvee_{a \leftarrow b_1,\ldots,b_n,\sim c_1,\ldots,\sim c_m \in \mathcal{R}} (\lambda_{b_1} \wedge \ldots \wedge \lambda_{b_n} \wedge \neg\lambda_{c_1} \wedge \ldots \wedge \neg\lambda_{c_m}) & \text{if } a \in \mathcal{B}_\mathcal{P} \backslash \mathcal{F} \end{cases}$$

should be replaced by

> **Definition 35** (Tc_P operator [Vlasselaer et al., 2015, 2016]). *Let $\mathcal{P}$ be a ground probabilistic logic program with probabilistic facts $\mathcal{F}$, rules $\mathcal{R}$ and atoms $\mathcal{B}_\mathcal{P}$. Let $\mathcal{I}$ be a parameterized interpretation with pairs (a, λ_a). Then, the Tc_P operator is $Tc_P(\mathcal{I}) = \{(a, \lambda_a) | a \in \mathcal{B}_\mathcal{P}\}$ where*
>
> $$\lambda'_a = \begin{cases} a & \text{if } a \in \mathcal{F} \\ \bigvee_{a \leftarrow b_1,\ldots,b_n,\sim c_1,\ldots,\sim c_m \in \mathcal{R}} (\lambda_{b_1} \wedge \ldots \wedge \lambda_{b_n} \wedge \neg\lambda_{c_1} \wedge \ldots \wedge \neg\lambda_{c_m}) & \text{if } a \in \mathcal{B}_\mathcal{P} \backslash \mathcal{F} \end{cases}$$

Page 177

The text:

> Vlasselaer et al. [2016] show that if each atom is selected frequently enough in step 1, then the same fixpoint $\mathrm{lfp}(Tc_P)$ 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

> Vlasselaer et al. [2016] show that if each atom is selected frequently enough in step 1, then the same fixpoint $\mathrm{lfp}(Tc_P)$ 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.

function EXACTSOLUTION($\mathcal{DT}$)
    $\text{ADD}_{tot}^{util} \leftarrow 0$
    for all $(u \rightarrow r) \in \mathcal{U}$ **do**
        Build $\text{BDD}(u)$, the BDD for u
        $\text{ADD}(u) \leftarrow$ PROBABILITYDD$(\text{BDD}_u(\mathcal{DT}))$
        $\text{ADD}^{util}(u) \leftarrow r \cdot \text{ADD}_u(\sigma)$
        $\text{ADD}_{tot}^{util} \leftarrow \text{ADD}_{tot}^{util} \oplus \text{ADD}^{util}(u)$
    end for
    let t_{max} be the terminal node of ADD_{tot}^{util} with the highest utility
    let p be a path from t_{max} to the root of ADD_{tot}^{util}
    return the Boolean decisions made on p
end function
function PROBABILITYDD(n)
    if n is the 1-terminal **then**
        return a 1-terminal
    end if
    if n is the 0-terminal **then**
        return a 0-terminal
    end if
    let h and l be the high and low children of n
    $\text{ADD}_h \leftarrow$ PROBABILITYDD(h)
    $\text{ADD}_l \leftarrow$ PROBABILITYDD(h)
    if n represents a decision d **then**
        return $\text{ITE}(d, \text{ADD}_h, \text{ADD}_l)$
    end if
    if n represents a fact with probability p **then**
        return $(p \cdot \text{ADD}_h) \oplus ((1-p) \cdot \text{ADD}_l)$
    end if
end function

should be replaced by

Algorithm 11 Function ExactSolution: Solving the DTProbLog decision problem exactly.

function ExactSolution($\mathcal{DT}$)
    $\mathrm{ADD}^{util}_{tot} \leftarrow 0$
    for all $(u \to r) \in \mathcal{U}$ **do**
        Build $\mathrm{BDD}(u)$, the BDD for u
        $\mathrm{ADD}(u) \leftarrow$ ProbabilityDD$(\mathrm{BDD}(u))$
        $\mathrm{ADD}^{util}(u) \leftarrow r \cdot \mathrm{ADD}(u)$
        $\mathrm{ADD}^{util}_{tot} \leftarrow \mathrm{ADD}^{util}_{tot} \oplus \mathrm{ADD}^{util}(u)$
    end for
    let t_{max} be the terminal node of $\mathrm{ADD}^{util}_{tot}$ with the highest utility
    let p be a path from t_{max} to the root of $\mathrm{ADD}^{util}_{tot}$
    return the Boolean decisions made on p
end function
function ProbabilityDD(n)
    if n is the 1-terminal **then**
        return a 1-terminal
    end if
    if n is the 0-terminal **then**
        return a 0-terminal
    end if
    let h and l be the high and low children of n
    $\mathrm{ADD}_h \leftarrow$ ProbabilityDD(h)
    $\mathrm{ADD}_l \leftarrow$ ProbabilityDD(l)
    if n represents a decision d **then**
        return $\mathrm{ITE}(d, \mathrm{ADD}_h, \mathrm{ADD}_l)$
    end if
    if n represents a fact with probability p **then**
        return $(p \cdot \mathrm{ADD}_h) \oplus ((1-p) \cdot \mathrm{ADD}_l)$
    end if
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 θ_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 $\mathbf{E}[c_{ik}|e]$ 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 $g(i)$. $\mathbf{E}[c_{ik}|e]$ is given by $\sum_{j\in g(i)} P(X_{ij} = x|e)$.

- Maximization: computes Π_{ik} for all $msw(i,x)$ and $k = 1,\ldots,n_i - 1$ as
$$\Pi_{ik} = \frac{\sum_{e\in E}\mathbf{E}[c_{ik}|e]}{\sum_{e\in E}\sum_{k=1}^{n_i}\mathbf{E}[c_{ik}|e]}$$

So, for each example e, X_{ij}s and x_{ik}s, 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_{ik} is number of times a variable X_{ij} takes value x_{ik} for any j, $\mathbf{E}[c_{ik}|e]$ is its expected value given example e. if $\mathbf{E}[c_{ik}]$ is its expected value given all the examples, then
$$\mathbf{E}[c_{ik}] = \sum_{t=1}^{T}\mathbf{E}[c_{ik}|e_t]$$
and
$$\Pi_{ik} = \frac{\mathbf{E}[c_{ik}]}{\sum_{k=1}^{n_i}\mathbf{E}[c_{ik}]}.$$

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 $msw(i,x)$ with domain D, with θ_j being a grounding substitution for i. Let $g(i)$ be the set of such substitutions:

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

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

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

So, for each example e, X_{ij}s and x_{ik}s, 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}, $\mathbf{E}[c_{ijk}|e]$ is its expected value given example e. If $\mathbf{E}[c_{ijk}]$ is its expected value given all the examples, then

$$\mathbf{E}[c_{ijk}] = \sum_{t=1}^{T}\mathbf{E}[c_{ijk}|e_t]$$

and

$$\Pi_{ijk} = \frac{\mathbf{E}[c_{ijk}]}{\sum_{k=1}^{n_i} \mathbf{E}[c_{ijk}]}.$$

Page 262-263, Algorithms 13-14

The text:

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

```
function PRISM-EM-NAIVE(E, 𝒫, ε)
    LL = −inf
    repeat
        LL_0 = LL
        for all i, k do                                   ▷ Expectation step
            E[c_ik] ← Σ_{e∈E} (Σ_{κ∈K_e, msw(i,x_ik)θ_j∈e} P(κ)) / P(e)
        end for
        for all i, k do                                   ▷ Maximization step
            Π_ik ← E[c_ik] / Σ_{k'=1}^{n_i} E[c_ik']
        end for
        LL ← Σ_{e∈E} log P(e)
    until LL − LL_0 < ε
    return LL, Π_ik for all i, k
end function
```

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

```
procedure GET-INSIDE-PROBS(q)
    for all i, k do
        P(msw(i, v_k)) ← Π_ik
    end for
    for i ← m → 1 do
        P(g_i) ← 0
        for j ← 1 → s_i do
            Let S_ij be h_ij1, ..., h_ijo
            P(g_i, S_ij) ← Π_{l=1}^{o} P(h_ijl)
            P(g_i) ← P(g_i) + P(g_i, S_ij)
        end for
    end for
end procedure
```

should be replaced by

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

function PRISM-EM-NAIVE($E, \mathcal{P}, \epsilon$)
  $LL = -inf$
  repeat
    $LL_0 = LL$
    for all i, j, k **do** ▷ Expectation step
      $\mathbf{E}[c_{ijk}] \leftarrow \sum_{e \in E} \frac{\sum_{\kappa \in K_e, msw(i\theta_j, x_{ik}) \in e} P(\kappa)}{P(e)}$
    end for
    for all i, j, k **do** ▷ Maximization step
      $\Pi_{ijk} \leftarrow \frac{\mathbf{E}[c_{ijk}]}{\sum_{k'=1}^{n_i} \mathbf{E}[c_{ijk'}]}$
    end for
    $LL \leftarrow \sum_{e \in E} \log P(e)$
  until $LL - LL_0 < \epsilon$
  return LL, Π_{ijk} for all i, j, k
end function

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

procedure GET-INSIDE-PROBS(q)
  for all i, j, k **do**
    $P(msw(i\theta_j, v_k)) \leftarrow \Pi_{ijk}$
  end for
  for $i \leftarrow m \rightarrow 1$ **do**
    $P(g_i) \leftarrow 0$
    for $j \leftarrow 1 \rightarrow s_i$ **do**
      Let S_{ij} be $h_{ij1}, \ldots, h_{ijo}$
      $P(g_i, S_{ij}) \leftarrow \prod_{l=1}^{o} P(h_{ijl})$
      $P(g_i) \leftarrow P(g_i) + P(g_i, S_{ij})$
    end for
  end for
end procedure

Pages 263-264

The text:

> If $g_i = msw(i, x_k)\theta_j$, 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 $msw(i, x_k)\theta_j$, 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 $g_i = msw(i, x_k)\theta_j$, 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

function PRISM-EM($E, \mathcal{P}, \epsilon$)
    $LL = -inf$
    repeat
        $LL_0 = LL$
        $LL =$ EXPECTATION(E)
        for all i **do**
            $Sum \leftarrow \sum_{k=1}^{n_i} \mathbf{E}[c_{ik}]$
            for $k = 1$ to n_i **do**
                $\Pi_{ik} = \frac{\mathbf{E}[c_{ik}]}{Sum}$
            end for
        end for
    until $LL - LL_0 < \epsilon$
    return LL, Π_{ik} for all i, k
end function

Algorithm 17 Procedure PRISM-EXPECTATION

function PRISM-EXPECTATION(E)
  $LL = 0$
  for all $e \in E$ **do**
    GET-INSIDE-PROBS(e)
    GET-OUTSIDE-PROBS(e)
    for all i **do**
      for $k = 1$ to n_i **do**
        $\mathbf{E}[c_{ik}] = \mathbf{E}[c_{ik}] + Q(msw(i, x_k))\Pi_{ik}/P(e)$
      end for
    end for
    $LL = LL + \log P(e)$
  end for
  return LL
end function

should be replaced by

Algorithm 16 Function PRISM-EM

function PRISM-EM($E, \mathcal{P}, \epsilon$)
  $LL = -inf$
  repeat
    $LL_0 = LL$
    $LL =$ EXPECTATION(E)
    for all i, j **do**
      $Sum \leftarrow \sum_{k=1}^{n_i} \mathbf{E}[c_{ijk}]$
      for $k = 1$ to n_i **do**
        $\Pi_{ijk} = \frac{\mathbf{E}[c_{ijk}]}{Sum}$
      end for
    end for
  until $LL - LL_0 < \epsilon$
  return LL, Π_{ijk} for all i, j, k
end function

Algorithm 17 Procedure PRISM-EXPECTATION

```
function PRISM-EXPECTATION(E)
    LL = 0
    for all e ∈ E do
        GET-INSIDE-PROBS(e)
        GET-OUTSIDE-PROBS(e)
        for all i, j do
            for k = 1 to n_i do
                E[c_ijk] = E[c_ijk] + Q(msw(iθ_j, x_k))Π_ijk/P(e)
            end for
        end for
        LL = LL + log P(e)
    end for
    return LL
end function
```

1 Page 272

The text:

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

should be replaced by

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

2 Page 281

The text:

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

should be replaced by

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

References

J. Vlasselaer, G. Van den Broeck, A. Kimmig, W. Meert, and L. De Raedt. Anytime inference in probabilistic logic programs with Tp-compilation. In *24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 1852–1858, 2015.

J. Vlasselaer, G. Van den Broeck, A. Kimmig, W. Meert, and L. De Raedt. Tp-compilation for inference in probabilistic logic programs. *International Journal of Approximate Reasoning*, 78:15–32, 2016. doi: 10.1016/j.ijar.2016.06.009.