Strudel: Learning Structured-Decomposable Probabilistic Circuits

Meihua Dang  
University of California, Los Angeles

Antonio Vergari  
University of California, Los Angeles

Guy Van den Broeck  
University of California, Los Angeles

September 23th, 2020 - Probabilistic Graphical Models (PGM)
Learning Structured-Decomposable Probabilistic Circuits

Probabilistic circuits (PCs) are \textit{tractable probabilistic models} \implies \textit{exact and efficient inference}!
Learning Structured-Decomposable Probabilistic Circuits

Probabilistic circuits (PCs) are **tractable probabilistic models**

⇒ **exact and efficient** inference!

known under many different names: **SPNs, PSDDs, CNets,**

⇒ checkout Guy’s tutorial tomorrow!
Learning **Structured-Decomposable** Probabilistic Circuits

To answer different classes of *tractable inferences*, PCs are required to have certain *structure properties*.

<table>
<thead>
<tr>
<th></th>
<th>EVI</th>
<th>MAR</th>
<th>CON</th>
<th>MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>smooth</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decomposable</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>deterministic</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

---

1Probabilistic circuits tutorial @ ECML-PKDD available at [https://www.youtube.com/watch?v=2RAG5-L9R70](https://www.youtube.com/watch?v=2RAG5-L9R70)
Learning **Structured-Decomposable** Probabilistic Circuits

To answer different classes of *tractable inferences*, PCs are required to have certain *structure properties* \(^1\)

<table>
<thead>
<tr>
<th></th>
<th>EVI</th>
<th>MAR</th>
<th>CON</th>
<th>MAP</th>
<th>KLD</th>
<th>Multiply</th>
<th>Expected Predictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>smooth</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decomposable</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>deterministic</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^1\)Probabilistic circuits tutorial @ ECML-PKDD available at https://www.youtube.com/watch?v=2RAG5-L9R70
Learning **Structured-Decomposable** Probabilistic Circuits

To answer different classes of *tractable inferences*, PCs are required to have certain *structure properties*.

<table>
<thead>
<tr>
<th></th>
<th>EVI</th>
<th>MAR</th>
<th>CON</th>
<th>MAP</th>
<th>KLD</th>
<th>Multiply</th>
<th>Expected Predictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>smooth</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>decomposable</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>deterministic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>struct.-decomp.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

For advanced queries...we need *structured-decomposable* PCs!

---

1. Probabilistic circuits tutorial @ ECML-PKDD available at https://www.youtube.com/watch?v=2RAG5-L9R70
Learning Structured-Decomposable Probabilistic Circuits

1. Initial structure
2. Generate candidate improvements
3. Score candidates \( \Rightarrow \) expensive!
4. go to 2.

LearnPSDD [Liang et al. 2017]
Learning Structured-Decomposable Probabilistic Circuits

1. Initial structure
2. Generate candidate improvements
3. Score candidates \(\Rightarrow\) expensive!
4. go to 2.

LearnPSDD [Liang et al. 2017]

1. Better initial structure
2. Generate single candidate \(\Rightarrow\) fast heuristics!
3. No scoring!
4. go to 2.

Strudel
Learning Structured-Decomposable Probabilistic Circuits

1. Initial structure
2. Generate candidate improvements
3. Score candidates \( \Rightarrow \) expensive!
4. go to 2.

LearnPSDD [Liang et al. 2017]

1. Better initial structure
2. Generate single candidate \( \Rightarrow \) fast heuristics!
3. No scoring!
4. go to 2.

Strudel

faster single structure learning \( \rightarrow \) fast mixture models!
Outline

Probabilistic Circuits

Scalable Learning Algorithm
  Structured Decomposability and Learning Algorithm
  Scale More with Mixtures

Advanced Queries

Conclusions
Probabilistic Circuits

Scalable Learning Algorithm
Structured Decomposability and Learning Algorithm
Scale More with Mixtures

Advanced Queries

Conclusions
Probabilistic Circuits (PCs)

PCs encode joint distributions via computational graphs.

\[
\begin{align*}
X_2 = 1 & \quad 0.9 \times 0.5 \times 0.4 \times 0.3 \\
X_2 = 0 & \quad 0.1 \times 0.5 \times 0.6 \times 0.7 \\
X_1 = 1 & \quad 0.5 \times 1.0 \times 0.3 \times 0.8 \\
X_1 = 0 & \quad 0.5 \times 1.0 \times 0.7 \times 0.6
\end{align*}
\]
Probabilistic Circuits (PCs)

PCs encode joint distributions via computational graphs

Input nodes are tractable distributions, e.g., indicator functions $p(X_i = 1) = [X_i = 1]$
Probabilistic Circuits (PCs)

PCs encode joint distributions via computational graphs

Product nodes are factorizations \( \prod_{c \in \text{in}(n)} P_c(x) \)
Probabilistic Circuits (PCs)

PCs encode joint distributions via computational graphs

Sum nodes are mixture models $\sum_{c \in \text{in}(n)} \theta_{n,c} p_c(x)$
Probabilistic Circuits

Scalable Learning Algorithm
  Structured Decomposability and Learning Algorithm
  Scale More with Mixtures

Advanced Queries

Conclusions
Structured-Decomposable PCs

Every product node over the same set of variables, decomposes in the same way...

...and all allowed ways to decompose are encoded in a vtree
Initialization

How to enforce structured-decomposability?

Start from some *simple but expressive* distribution, then *compile* it into a PC with a vtree

Q: which distribution?

A: Use a *Chow-Liu tree* as the “best” initial PC

- *easy* compilation to structured-decomposable PCs
- *best tree model* according to KLD [Chow et al. 1968]
Structure refinement

How to improve the current structure?

Q: How to build a more expressive PC...while preserving properties?
**Structure refinement**

How to improve the current structure?

**Q:** How to build a *more expressive PC*...while *preserving properties*?

**A:** by applying the *split operator*!
How to split?

How to pick an **edge** to split?

⇒ *take the edge with most samples “flowing” through it*

How to pick a **variable** to condition on?

⇒ *take the variable with the strongest dependencies among others*
**Strudel: accurate PCs**

Now we have good single model
Fit the data better and scale more: *mixture models*

<table>
<thead>
<tr>
<th>DATASET</th>
<th>LEARNPSDD</th>
<th>STRUDEL</th>
<th>SELSPN</th>
<th>SEASP0N</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLTC5</td>
<td>-6.03</td>
<td>-6.06</td>
<td>-6.03↓</td>
<td>-6.07†</td>
</tr>
<tr>
<td>MSNBC</td>
<td>-6.04</td>
<td>-6.05</td>
<td>-6.04↓</td>
<td>-6.06†</td>
</tr>
<tr>
<td>KDD</td>
<td>-2.17</td>
<td>-2.17</td>
<td>-2.16↓</td>
<td>-2.16†</td>
</tr>
<tr>
<td>AUDIO</td>
<td>-41.51</td>
<td>-42.26</td>
<td>-41.23↓</td>
<td>-40.13↓</td>
</tr>
<tr>
<td>JESTER</td>
<td>-54.63</td>
<td>-55.30</td>
<td>-54.38↓</td>
<td>-53.08↓</td>
</tr>
<tr>
<td>NETFLIX</td>
<td>-58.53</td>
<td>-58.68</td>
<td>-57.98↓</td>
<td>-56.91↓</td>
</tr>
<tr>
<td>ACCIDENTS</td>
<td>-28.291</td>
<td>-29.46</td>
<td>-26.88↓</td>
<td>-30.02↓</td>
</tr>
<tr>
<td>RETAIL</td>
<td>-10.92</td>
<td>-10.90</td>
<td>-10.88↓</td>
<td>-10.97†</td>
</tr>
<tr>
<td>DNA</td>
<td>-83.02</td>
<td>-87.10</td>
<td>-80.44↓</td>
<td>-81.76↓</td>
</tr>
<tr>
<td>KOSAREK</td>
<td>-10.99</td>
<td>-10.98</td>
<td>-10.85↓</td>
<td>-11.00↓</td>
</tr>
<tr>
<td>MSWEB</td>
<td>-9.93</td>
<td>-10.19</td>
<td>-9.93↓</td>
<td>-10.25↓</td>
</tr>
<tr>
<td>BOOK</td>
<td>-36.06</td>
<td>-35.77</td>
<td>-36.01↓</td>
<td>-34.91↓</td>
</tr>
<tr>
<td>EACHMOVIE</td>
<td>-55.41</td>
<td>-59.47</td>
<td>-55.73↓</td>
<td>-53.28↓</td>
</tr>
<tr>
<td>WEBKB</td>
<td>-161.42</td>
<td>-160.50</td>
<td>-158.52↓</td>
<td>-157.88↓</td>
</tr>
<tr>
<td>ROUTERS-52</td>
<td>-93.30</td>
<td>-92.38</td>
<td>-88.48↓</td>
<td>-86.38↓</td>
</tr>
<tr>
<td>20NEWS-GRP</td>
<td>-160.43</td>
<td>-160.77</td>
<td>-158.68↓</td>
<td>-153.63↓</td>
</tr>
<tr>
<td>BBC</td>
<td>-260.24</td>
<td>-258.96</td>
<td>-259.35↑</td>
<td>-253.13↓</td>
</tr>
<tr>
<td>AD</td>
<td>-20.13</td>
<td>-16.52</td>
<td>-16.94↑</td>
<td>-16.77†</td>
</tr>
</tbody>
</table>
Probabilistic Circuits

Scalable Learning Algorithm
  Structured Decomposability and Learning Algorithm
  Scale More with Mixtures

Advanced Queries

Conclusions
Strudel: Mixtures of PCs

Difficulties:

- **Structured-decomposable**
  - *each component* being struct.-decomp....conform to the *same vtree*

- **Scalable**
  - memory, evaluation, learning...
**Strudel: Mixtures of PCs**

Solution: learn the mixtures *sharing the same structure*, but having different parameters

- Simple learning and memory efficiency
- Computational efficiency by *circuit flows!*

$$\log\text{sumexp}(f_{\mathcal{M}}(x)^T \cdot \log(\Theta) + \log(w))$$

E.g., #component 100; #paras each 10,000; N 100,000:

$$(100,000, 10,000)^* (10,000, 100) + (1, 100)$$

⇒ stop by at poster session :)
Probabilistic Circuits

Scalable Learning Algorithm
  Structured Decomposability and Learning Algorithm
  Scale More with Mixtures

Advanced Queries

Conclusions
Advanced Queries

Compute the *expected predictions* [Khosravi et al. 2019]

⇒ *single* probabilistic circuits **perform better** than the baseline, the **mixtures** also help further **reduce the error**
Conclusions

Probabilistic Circuits can answer advanced queries

⇒ structured-decomposable PCs

We propose a simple and scalable structure learning algorithm

Scale learning:

- Fast heuristic
- Cheap mixtures of circuits sharing the structures
- Circuit flows


Technical Details: Determinism and Circuit Flows

Determinism: for every sum node, at most one of its input is non-zero

**Circuit flows**: encodes which parameters are activated by different input configurations

Benefits:

- Closed-form MLE
- Fast inference \( f_c(x)^T \cdot \log(\theta) \)

\[
\begin{bmatrix}
0 & 1 & 0 & 0 & 1 & 0 & \ldots \\
\end{bmatrix} \cdot 
\log\left(\begin{bmatrix}
.4 & .6 & .3 & .7 & .8 & .2 & \ldots \\
\end{bmatrix}\right)
\]

Figure: For input configuration 
\(X_1 = 1, X_2 = 0, X_3 = 1, X_4 = 0\),
red colors indicate all the active edges