Logic & Probabilistic Circuits

Guy Van den Broeck  
University of California, Los Angeles  
guyvdb@cs.ucla.edu

YooJung Choi  
Arizona State University  
yj.choi@asu.edu

August 22nd, 2023 - Logic and Algorithms in Database Theory and AI Boot Camp @ Simons Institute
Objective

Circuits are an assembly language for tractable logic and probabilistic reasoning

Even though logic is central to this Simons program, we will couch this tutorial in probability...

Most AI and DB interest in tractable logic circuits for the past 15 years has been as a means of doing probabilistic inference

Much richer query languages <3

We live in the age of probabilistic generative AI... :-)

We will spare you most of the machine learning details, and instead focus on representations, query languages, reasoning algorithms, and connections to theory.
Objective

- Circuits are an assembly language for tractable logic and probabilistic reasoning
- Even though logic is central to this Simons program, we will couch this tutorial in probability...
  - Most AI and DB interest in tractable logic circuits for the past 15 years has been as a means of doing probabilistic inference
  - Much richer query languages <3
  - We live in the age of probabilistic generative AI... 😊
- We will spare you most of the machine learning details, and instead focus on representations, query languages, reasoning algorithms, and connections to theory.
Circuits are an assembly language for tractable logic and probabilistic reasoning.

Even though logic is central to this Simons program, we will couch this tutorial in probability...

- Most AI and DB interest in tractable logic circuits for the past 15 years has been as a means of doing probabilistic inference.
- Much richer query languages <3
- We live in the age of probabilistic generative AI... ☺️

We will spare you most of the machine learning details, and instead focus on representations, query languages, reasoning algorithms, and connections to theory.
Questions to be answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Questions to be answered today

1. **What are probabilistic queries? Are current models tractable?** (Guy)
2. **What are probabilistic circuits and why are they tractable?** (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Questions to be answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Questions to be answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Questions to be answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Questions to be answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Questions to be answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Questions to be answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Acknowledgements

This tutorial is based on our (joint) tutorials and slides from

- Antonio Vergari
- Robert Peharz
- Nicola Di Mauro
- Honghua Zhang
- Benjie Wang
The Alphabet Soup of probabilistic models
Fully factorized

NaiveBayes  AndOrGraphs  PDGs

Trees  PSDDs  CNets  LTM  SPNs  NADEs

Thin Junction Trees  ACs  MADEs  MAFs  VAEs

DPPs  FVSBNs  TACs  IAFs  NAFs  RAEs

Mixtures  BNs  NICE  FGs  GANs

RealNVP  MNs  GPT

Intractable and tractable models
tractability is a spectrum
Expressive models without compromises
a \textit{unifying framework} for tractable models
Why tractable inference?

or the inherent trade-off of tractability vs. expressiveness
Why probabilistic inference?

$q_1$: What is the probability that today is a Monday and there is a traffic jam on Westwood Blvd.?
Why probabilistic inference?

$q_1$: What is the probability that today is a Monday and there is a traffic jam on Westwood Blvd.?

$X = \{\text{Day, Time, Jam}_{\text{str1}}, \text{Jam}_{\text{str2}}, \ldots, \text{Jam}_{\text{strN}}\}$

$q_1(m) = p_m(\text{Day} = \text{Mon}, \text{Jam}_{\text{Wwood}} = 1)$
Why probabilistic inference?

$q_1$: What is the probability that today is a Monday and there is a traffic jam on Westwood Blvd.?

\[ X = \{\text{Day, Time, Jam}_{\text{Str1}}, \text{Jam}_{\text{Str2}}, \ldots, \text{Jam}_{\text{StrN}}\} \]

\[ q_1(m) = p_m(\text{Day} = \text{Mon}, \text{Jam}_{\text{Westwood}} = 1) \]

\[ \implies \text{marginals} \]
Why probabilistic inference?

$q_2$: Which day is most likely to have a traffic jam on my route to campus?

$X = \{\text{Day, Time, Jam}_{\text{Str1}}, \text{Jam}_{\text{Str2}}, \ldots, \text{Jam}_{\text{StrN}}\}$

$q_2(m) = \arg\max_d p_m(\text{Day} = d \land \bigvee_{i \in \text{route}} \text{Jam}_{\text{Str}_i})$
Why probabilistic inference?

$q_2$: Which day is most likely to have a traffic jam on my route to campus?

$X = \{\text{Day}, \text{Time}, \text{Jam}_{\text{Str}_1}, \text{Jam}_{\text{Str}_2}, \ldots, \text{Jam}_{\text{Str}_N}\}$

$q_2(m) = \arg \max_d p_m(\text{Day} = d \land \bigvee_{i \in \text{route}} \text{Jam}_{\text{Str}_i})$

$\Rightarrow$ marginals + MAP + logical events
Tractable Probabilistic Inference

A class of queries $Q$ is tractable on a family of probabilistic models $M$ iff for any query $q \in Q$ and model $m \in M$ exactly computing $q(m)$ runs in time $O(poly(|m|))$. 
A class of queries $Q$ is tractable on a family of probabilistic models $\mathcal{M}$ iff for any query $q \in Q$ and model $m \in \mathcal{M}$, exactly computing $q(m)$ runs in time $O(\text{poly}(|m|))$. 

$\Rightarrow$ often poly will in fact be linear!
tractable bands
q3: What is the probability that today is a Monday at 12.00 and there is a traffic jam only on Westwood Blvd.?
Complete evidence (EVI)

$q_3$: What is the probability that today is a Monday at 12.00 and there is a traffic jam only on Westwood Blvd.?

$X = \{\text{Day, Time, } \text{Jam}_{\text{Wwood}}, \text{Jam}_{\text{Str2}}, \ldots, \text{Jam}_{\text{StrN}}\}$

$q_3(m) = p_m(X = \{\text{Mon, 12.00, 1, 0, \ldots, 0}\})$
**Complete evidence (EVI)**

**q₃**: What is the probability that today is a Monday at 12.00 and there is a traffic jam only on Westwood Blvd.?

\[ X = \{\text{Day}, \text{Time}, \text{Jam}_{\text{Wood}}, \text{Jam}_{\text{Str2}}, \ldots, \text{Jam}_{\text{StrN}}\} \]

\[ q₃(m) = p_m(X = \{\text{Mon, 12.00, 1, 0, } \ldots, 0\}) \]

...fundamental in **maximum likelihood learning**

\[ \theta_{\text{MLE}}^m = \arg\max_{\theta} \prod_{x \in D} p_m(x; \theta) \]
Generative Adversarial Networks

$$\min_\theta \max_\phi \mathbb{E}_{x \sim p_{\text{data}}(x)} \left[ \log D_\phi(x) \right] + \mathbb{E}_{z \sim p(z)} \left[ \log(1 - D_\phi(G_\theta(z))) \right]$$

\[ \text{Goodfellow et al., “Generative adversarial nets”, 2014} \]
min_θ \max_ϕ \mathbb{E}_{x \sim p_{data}(x)} \left[ \log D_ϕ(x) \right] + \mathbb{E}_{z \sim p(z)} \left[ \log (1 - D_ϕ(G_θ(z))) \right]

- no explicit likelihood!
  \implies \text{adversarial training instead of MLE}

- good sample quality
  \implies \text{but lots of samples needed for MC}

- unstable training
  \implies \text{mode collapse}

---

Goodfellow et al., “Generative adversarial nets”, 2014
tractable bands
Variational Autoencoders

\[ p_\theta(x) = \int p_\theta(x \mid z)p(z)dz \]

an explicit likelihood model!

Rezende et al., “Stochastic backprop. and approximate inference in deep generative models”, 2014
Kingma and Welling, “Auto-Encoding Variational Bayes”, 2014
Variational Autoencoders

$$\log p_\theta(x) \geq \mathbb{E}_{z \sim q_\phi(z|x)} \left[ \log p_\theta(x \mid z) \right] - \text{KL}(q_\phi(z \mid x) \mid \mid p(z))$$

- an explicit likelihood model!
- ... but computing $\log p_\theta(x)$ is intractable
  - $\Rightarrow$ an infinite and uncountable mixture
  - $\Rightarrow$ no tractable EVI
- we need to optimize the ELBO...
  - $\Rightarrow$ which is “tricky”
<table>
<thead>
<tr>
<th>$\mathcal{M}$</th>
<th>$Q$:</th>
<th>$\text{EVI}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GANs</td>
<td>$\times$</td>
<td></td>
</tr>
<tr>
<td>VAEs</td>
<td>$\times$</td>
<td></td>
</tr>
</tbody>
</table>

*tractable bands*
Normalizing flows

$$p_X(x) = p_Z(f^{-1}(x)) \left| \det \left( \frac{\delta f^{-1}}{\delta x} \right) \right|$$

- an explicit likelihood!
  $\Rightarrow$ tractable EVI queries!

- many neural variants
  - RealNVP \((Dinh \ et \ al. \ 2016)\),
  - MAF \((Papamakarios \ et \ al. \ 2017)\)
  - MADE \((Germain \ et \ al. \ 2015)\),
  - PixelRNN \((Oord \ et \ al. \ 2016)\)
Normalizing flows

\[ p_X(x) = p_Z(f^{-1}(x)) \left| \det \left( \frac{\delta f^{-1}}{\delta x} \right) \right| \]

- an explicit likelihood!
- tractable EVI queries!
- many neural variants
  - RealNVP (Dinh et al. 2016),
  - MAF (Papamakarios et al. 2017)
  - MADE (Germain et al. 2015),
  - PixelRNN (Oord et al. 2016)
Marginal queries (MAR)

$q_1$: What is the probability that today is a Monday at 12:00 and there is a traffic jam only on Westwood Blvd.?
Marginal queries (MAR)

$q_1$: What is the probability that today is a Monday at 12.00 and there is a traffic jam only on Westwood Blvd.?

$q_1(m) = p_m(\text{Day} = \text{Mon, Jam}_{\text{Westwood}} = 1)$
Marginal queries (MAR)

$q_1$: What is the probability that today is a Monday at 12.00 and there is a traffic jam only on Westwood Blvd.?

$q_1(m) = p_m(\text{Day} = \text{Mon}, \text{Jam}_{\text{Westwood}} = 1)$

General: $p_m(e) = \int p_m(e, H) dH$

where $E \subset X$, $H = X \setminus E$
**Marginal queries (MAR)**

$q_1$: *What is the probability that today is a Monday at 12:00 and there is a traffic jam only on Westwood Blvd.?*

$q_1(m) = p_m(\text{Day} = \text{Mon}, \text{Jam}_{\text{Westwood}} = 1)$

tractable MAR $\implies$ tractable *conditional queries* (CON):

$$p_m(q \mid e) = \frac{p_m(q, e)}{p_m(e)}$$
Tractable MAR: scene understanding

Fast and exact marginalization over unseen or “do not care” parts in the scene

Stelzner et al., “Faster Attend-Infer-Repeat with Tractable Probabilistic Models”, 2019
Normalizing flows

\[ p_X(x) = p_Z(f^{-1}(x)) \left| \det \left( \frac{\delta f^{-1}}{\delta x} \right) \right| \]

an explicit likelihood!

⇒ tractable EVI queries!
Normalizing flows

\[ p_X(x) = p_Z(f^{-1}(x)) \left| \det \left( \frac{\delta f^{-1}}{\delta x} \right) \right| \]

- an explicit likelihood!
  \[ \Rightarrow \text{tractable EVI queries!} \]
- **MAR is generally intractable:**
  we cannot easily integrate over high-dimensional \( f \)
tractable bands
**Probabilistic Graphical Models (PGMs)**

*Declarative semantics:* a clean separation of modeling assumptions from inference

**Nodes:** random variables

**Edges:** dependencies

**Inference:**
- conditioning (*Darwiche* 2001; *Sang et al.* 2005)
- elimination (*Zhang et al.* 1994; *Dechter* 1998)
- message passing (*Yedidja et al.* 2001; *Dechter et al.* 2002; *Choi et al.* 2010; *Sontag et al.* 2011)
Complexity of MAR on PGMs

**Exact complexity:** Computing MAR and CON is \#P-hard

\[ \Rightarrow \quad (\text{Cooper 1990; Roth 1996}) \]

**Approximation complexity:** Computing MAR and CON approximately within a relative error of \(2^{n^{1-\varepsilon}}\) for any fixed \(\varepsilon\) is NP-hard

\[ \Rightarrow \quad (\text{Dagum et al. 1993; Roth 1996}) \]
Treewidth:
Informally, how tree-like is the graphical model $m$?

**Fixed-parameter tractable:** MAR and CON on a graphical model $m$ with treewidth $w$ take time $O(|X| \cdot 2^w)$ (Dechter 1998; Koller et al. 2009).

⇒ what about bounding the treewidth by design?
Low-treewidth PGMs

*Trees*  
(Meilă et al. 2000)

*Polytrees*  
(Dasgupta 1999)

*Thin Junction trees*  
(Bach et al. 2001)

If treewidth is bounded (e.g. $\simeq 20$), exact MAR and CON inference is possible in practice
Tree distributions

A tree-structured BN (Meilă et al. 2000) where each $X_i \in X$ has at most one parent $\text{Pa}_{X_i}$.

$$p(X) = \prod_{i=1}^{n} p(x_i|\text{Pa}_{x_i})$$

**Exact querying:** EVI, MAR, CON tasks linear for trees: $O(|X|)$

**Exact learning** from $d$ examples takes $O(|X|^2 \cdot d)$ with the classical Chow-Liu algorithm

1 Chow et al., “Approximating discrete probability distributions with dependence trees”, 1968
tractable bands
What do we lose?

*Expressiveness*: Ability to represent rich and complex classes of distributions

Bounded-treewidth PGMs lose the ability to represent *all possible distributions* ...

Mixtures as a convex combination of $k$ (simpler) probabilistic models

$$p(X) = w_1 \cdot p_1(X) + w_2 \cdot p_2(X)$$

EVI, MAR, CON queries scale linearly in $k$
Mixtures

Mixtures as a convex combination of $k$ (simpler) probabilistic models

\[ p(X) = p(Z = 1) \cdot p_1(X|Z = 1) + p(Z = 2) \cdot p_2(X|Z = 2) \]

Mixtures are marginalizing a **categorical latent variable** $Z$ with $k$ values

$\Rightarrow$ **increased expressiveness**
Expressiveness and efficiency

Expressiveness: Ability to represent rich and effective classes of functions

⇒ mixture of Gaussians can approximate any probability density!

Expressiveness and efficiency

**Expressiveness**: Ability to represent rich and effective classes of functions

⇒ mixture of Gaussians can approximate any probability density!

**Expressive efficiency (aka Succinctness)**:
Ability to represent rich and effective classes of functions **compactly**

⇒ but how many components does a Gaussian mixture need?

---

How expressive efficient are mixtures?
How expressive efficient are mixtures?
How expressive efficient are mixtures?
How expressive efficient are mixtures?
How expressive efficient are mixtures?
How expressive efficient are mixtures?
How expressive efficient are mixtures?
How expressive efficient are mixtures?

solution: deep mixtures as in deep generative models
<table>
<thead>
<tr>
<th>$\mathcal{M}$</th>
<th>$Q$: EVI</th>
<th>MAR</th>
<th>CON</th>
</tr>
</thead>
<tbody>
<tr>
<td>GANs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAEs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flows</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trees</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mixtures</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

tractable bands
Maximum A Posteriori (MAP)
aka Most Probable Explanation (MPE)

$q_5$: Which combination of roads is most likely to be jammed on Monday at 9am?
Maximum A Posteriori (MAP)

aka Most Probable Explanation (MPE)

$q_5$: Which combination of roads is most likely to be jammed on Monday at 9am?

$q_5(m) = \arg\max_j p_m(j_1, j_2, \ldots | \text{Day} = M, \text{Time} = 9)$
Maximum A Posteriori (MAP)

aka Most Probable Explanation (MPE)

$q_5$: Which combination of roads is most likely to be jammed on Monday at 9am?

$q_5(m) = \arg\max_j p_m(j_1, j_2, \ldots | \text{Day} = M, \text{Time} = 9)$

General: $\arg\max_q p_m(q | e)$

where $Q \cup E = X$
Maximum A Posteriori (MAP)

aka Most Probable Explanation (MPE)

$q_5$: Which combination of roads is most likely to be jammed on Monday at 9am?

...intractable for latent variable models!

$$\max_q p_m(q \mid e) = \max_q \sum_z p_m(q, z \mid e)$$

$$\neq \sum_z \max_q p_m(q, z \mid e)$$
MAP inference: image inpainting

Predicting \textit{arbitrary patches} given a \textit{single} model without the need of retraining.

\underline{Poon and Domingos, “Sum-Product Networks: a New Deep Architecture”, 2011}
\underline{Sguerra and Cozman, “Image classification using sum-product networks for autonomous flight of micro aerial vehicles”, 2016}
tractable bands
Marginal MAP (MMAP)
aka Bayesian Network MAP

Q: Which combination of roads is most likely to be jammed on Monday at 9am?
Marginal MAP (MMAP)
aka Bayesian Network MAP

$q_6$: Which combination of roads is most likely to be jammed on Monday at 9am?

$q_6(m) = \arg\max_j p_m(j_1, j_2, \ldots | \text{Time = 9})$
Marginal MAP (MMAP)
aka Bayesian Network MAP

$q_6$: Which combination of roads is most likely to be jammed on Monday at 9am?

$q_6(m) = \arg\max j p_m(j_1, j_2, \ldots | \text{Time} = 9)$

General: $\arg\max_q p_m(q | e)$

$= \arg\max_q \sum_h p_m(q, h | e)$

where $Q \cup H \cup E = X$
Marginal MAP (MMAP) aka Bayesian Network MAP

$q_6$: Which combination of roads is most likely to be jammed on Monday at 9am?

$q_6(m) = \arg\max_j p_m(j_1, j_2, \ldots | \text{Time} = 9)$

$\Rightarrow$ NP²P-complete \((Park \ et \ al. \ 2006)\)

$\Rightarrow$ NP-hard for trees \((de \ Campos \ 2011)\)

$\Rightarrow$ NP-hard even for Naive Bayes \((ibid.)\)
<table>
<thead>
<tr>
<th>(M)</th>
<th>(Q:)</th>
<th>EVI</th>
<th>MAR</th>
<th>CON</th>
<th>MAP</th>
<th>MMAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>GANs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAEs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flows</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trees</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mixtures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**tractable bands**
q2: Which day is most likely to have a traffic jam on my route to campus?
Advanced queries

$q_2$: Which day is most likely to have a traffic jam on my route to campus?

$q_2(m) = \arg\max_d p_m(Day = d \land \bigvee_{i \in \text{route}} \text{Jam}_{\text{Str} i})$

$\implies$ marginals + MAP + logical events
**Advanced queries**

$q_2$: Which day is most likely to have a traffic jam on my route to campus?

$q_7$: What is the probability of seeing more traffic jams in Westwood than Hollywood?

---

Advanced queries

$q_2$: Which day is most likely to have a traffic jam on my route to campus?

$q_7$: What is the probability of seeing more traffic jams in Westwood than Hollywood?

⇒ counts + group comparison
Advanced queries

q2: Which day is most likely to have a traffic jam on my route to campus?

q7: What is the probability of seeing more traffic jams in Westwood than Hollywood?

q8: Is traffic more uncertain on weekdays?
Advanced queries

\( q_2 \): Which day is most likely to have a traffic jam on my route to campus?

\( q_7 \): What is the probability of seeing more traffic jams in Westwood than Hollywood?

\( q_8 \): Is traffic more uncertain on weekdays?

⇒ information-theoretic queries
Advanced queries

q2: Which day is most likely to have a traffic jam on my route to campus?

q7: What is the probability of seeing more traffic jams in Westwood than Hollywood?

q8: Is traffic more uncertain on weekdays?

q9: What is the causal effect of doing road works?
Advanced queries

q2: Which day is most likely to have a traffic jam on my route to campus?

q7: What is the probability of seeing more traffic jams in Westwood than Hollywood?

q8: Is traffic more uncertain on weekdays?

q9: What is the causal effect of doing road works?

⇒ causal backdoor estimation
tractable bands
<table>
<thead>
<tr>
<th>(\mathcal{M})</th>
<th>(Q:)</th>
<th>EVI</th>
<th>MAR</th>
<th>CON</th>
<th>MAP</th>
<th>MMAP</th>
<th>ADV</th>
</tr>
</thead>
<tbody>
<tr>
<td>GANs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAEs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flows</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trees</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mixtures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**tractable bands**
Fully factorized models

A completely disconnected graph. Example: Product of Bernoullis (PoBs)

\[
p(x) = \prod_{i=1}^{n} p(x_i)
\]

Complete evidence, marginals and MAP, MMAP inference is \textit{linear}!

\[\Rightarrow \quad \text{but definitely not expressive...}\]
tractable bands
Expressive models are not very tractable...
and **tractable** ones are not very expressive...
probabilistic circuits are at the “sweet spot”
1. *What are probabilistic queries? Are current models tractable?* (Guy)
2. *What are probabilistic circuits and why are they tractable?* (Guy)
3. *What is the connection to logical circuit languages?* (YooJung)
4. *How do I compile my favorite model into a circuit?* (YooJung)
5. *How are circuit size and tractability related?* (YooJung)
6. *What’s the most impressive query we can efficiently compute?* (YooJung)
7. *Are all tractable distributions probabilistic circuits?* (Guy)
8. *How to learn probabilistic circuits from data?* (Guy)
Questions answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Probabilistic Circuits
Goal

Given a reasoning task can we design a class of expressive models that is tractable for it?
Goal

Given a reasoning task, can we design a class of deep computational graphs that is tractable for it?
more tractable

less tractable

more expressive

less expressive
Expressive models are not very tractable...
Tractable models are not that expressive...
Circuits can be both expressive and tractable!
Start simple...
then make it more expressive!
impose structure!
**Input distributions**
as computational nodes

**Base case:** a single node encoding a distribution

⇒ e.g., Gaussian PDF continuous random variable
Input distributions

as computational nodes

Base case: a single node encoding a distribution

⇒ e.g., indicators for $X$ or $\neg X$ for Boolean random variable
Input distributions

as computational nodes

Simple distributions are tractable “black boxes” for:

- EVI: output $p(x)$ (density or mass)
- MAR: output 1 (normalized) or $Z$ (unnormalized)
- MAP: output the mode
Mixture models

as computational graphs

\[ p(X) = w_1 \cdot p_1(X_1) + w_2 \cdot p_2(X_1) \]

⇒ translating inference to data structures...
Mixture models

as computational graphs

\[ p(X_1) = 0.2 \cdot p_1(X_1) + 0.8 \cdot p_2(X_1) \]

...e.g., as a weighted sum unit over Gaussian input distributions
Mixture models
as computational graphs

\[ p(X = 5) = 0.2 \cdot p_1(X_1 = 5) + 0.8 \cdot p_2(X_1 = 5) \]

⇒ inference = feedforward evaluation
Mixture models

as computational graphs

A simplified notation:

⇒ scopes attached to inputs
⇒ edge directions omitted
Factorizations

as computational graphs

\[ p(X_1, X_2, X_3) = p(X_1) \cdot p(X_2) \cdot p(X_3) \]

⇒ e.g. modeling a multivariate Gaussian with diagonal covariance matrix...
Factorizations as computational graphs

\[ p(X_1, X_2, X_3) = p(X_1) \cdot p(X_2) \cdot p(X_3) \]

\[ X_1 \quad X_2 \quad X_3 \]

\[ \Rightarrow \quad \text{...with a product node over some univariate Gaussian distribution} \]
Factorizations as computational graphs

\[ p(x_1, x_2, x_3) = p(x_1) \cdot p(x_2) \cdot p(x_3) \]
Factorizations as computational graphs

\[ p(x_1, x_2, x_3) = p(x_1) \cdot p(x_2) \cdot p(x_3) \]

\[ \Rightarrow \text{feedforward evaluation} \]
A grammar for tractable models

Recursive semantics of probabilistic circuits
A grammar for tractable models

Recursive semantics of probabilistic circuits
A grammar for tractable models

Recursive semantics of probabilistic circuits
A grammar for tractable models
Recursive semantics of probabilistic circuits
A grammar for tractable models

Recursive semantics of probabilistic circuits
Building PCs in Python with SPFlow

```python
import spn.structure.leaves.parametric.Parametric as param
from param import Categorical, Gaussian

PC = 0.4 * (Categorical(p=[0.2, 0.8], scope=0) *
    (0.3 * (Gaussian(mean=1.0, stddev=1.0, scope=1) *
        Categorical(p=[0.4, 0.6], scope=2))
    + 0.7 * (Gaussian(mean=-1.0, stddev=1.0, scope=1) *
        Categorical(p=[0.6, 0.4], scope=2)))
    + 0.6 * (Categorical(p=[0.2, 0.8], scope=0) *
        Gaussian(mean=0.0, stddev=0.1, scope=1) *
        Categorical(p=[0.4, 0.6], scope=2))
```

EVI queries = feedforward evaluation

\[ p(X_1 = -1.85, X_2 = 0.5, X_3 = -1.3, X_4 = 0.2) \]
\[ p(X_1 = -1.85, X_2 = 0.5, X_3 = -1.3, X_4 = 0.2) \]
EVI queries = feedforward evaluation

\[ p(X_1 = -1.85, X_2 = 0.5, X_3 = -1.3, X_4 = 0.2) = 0.75 \]
Just sum, products and distributions?

just arbitrarily compose them like a neural network!
Just sum, products and distributions?

Just arbitrarily compose them like a neural network!

⇒ structural properties needed for tractability
Which structural constraints ensure tractability?
**Decomposability**

A product node is decomposable if its children depend on disjoint sets of variables

\[ \Rightarrow \text{just like in factorization!} \]

A decomposable circuit is shown on the left, and a non-decomposable circuit on the right.

\[ \times \]

\[ \text{decomposable circuit} \]

\[ \times \]

\[ \text{non-decomposable circuit} \]

*Darwiche and Marquis, “A knowledge compilation map”, 2002*
**Smoothness**

*aka completeness*

A sum node is smooth if its children depend of the same variable sets

\[ \Rightarrow \quad \text{otherwise not accounting for some variables} \]

\[ \Rightarrow \quad \text{smoothness can be easily enforced} \quad (\text{Shih et al. 2019}) \]

---

Darwiche and Marquis, “A knowledge compilation map”, 2002
Smoothness + decomposability = tractable MAR

Computing arbitrary integrations (or summations) ⇒ linear in circuit size!

E.g., suppose we want to compute $Z$
(the distribution’s normalizing constant):

$$\int p(x)dx$$
Smoothness + decomposability = tractable MAR

If \( p(x) = \sum_i w_i p_i(x) \), (smoothness):

\[
\int p(x) \, dx = \int \sum_i w_i p_i(x) \, dx =
\]

\[
= \sum_i w_i \int p_i(x) \, dx
\]

⇒ integrals are “pushed down” to children
Smoothness + decomposability = tractable MAR

If \( p(x, y, z) = p(x)p(y)p(z) \), (decomposability):

\[
\int \int \int p(x, y, z) \, dx \, dy \, dz = \\
= \int \int \int p(x)p(y)p(z) \, dx \, dy \, dz = \\
= \int p(x) \, dx \int p(y) \, dy \int p(z) \, dz
\]

\[\Rightarrow \text{integrals decompose into easier ones}\]
**Smoothness** + **decomposability** = **tractable MAR**

Forward pass evaluation for MAR

⇒ *linear in circuit size!*

E.g. to compute $p(x_2, x_4)$:

1. leafs over $X_1$ and $X_3$ output $Z_i = \int p(x_i) dx_i$
   ⇒ for normalized leaf distributions: 1.0
2. leafs over $X_2$ and $X_4$ output **EVI**
3. feedforward evaluation (bottom-up)
**Smoothness** + **decomposability** = **tractable MAR**

Forward pass evaluation for MAR

⇒ *linear in circuit size!*

E.g. to compute $p(x_2, x_4)$:

- Leaf nodes over $X_1$ and $X_3$ output $Z_i = \int p(x_i) dx_i$
  ⇒ *for normalized leaf distributions:* **1.0**

- Leaf nodes over $X_2$ and $X_4$ output **EVI**

- Feedforward evaluation (bottom-up)
**Smoothness** + **decomposability** = **tractable MAR**

Forward pass evaluation for MAR

⇒ *linear in circuit size!*

E.g. to compute $p(x_2, x_4)$:
- Leafs over $X_1$ and $X_3$ output $Z_i = \int p(x_i) \, dx_i$
  ⇒ *for normalized leaf distributions: 1.0*
- Leafs over $X_2$ and $X_4$ output **EVI**
- Feedforward evaluation (bottom-up)
Tractable MAR

\[ \text{EVI} \quad 10,958.72 \text{ nats} \]
\[ \text{MAR} \quad 5,387.55 \text{ nats} \]

---

Peharz et al., “Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits”, 2020
\[ p(q \mid e) = \frac{p(q, e)}{p(e)} \]

1. evaluate \( p(q, e) \) \( \Rightarrow \) one feedforward pass
2. evaluate \( p(e) \) \( \Rightarrow \) another feedforward pass
   \( \Rightarrow \) ...still linear in circuit size!
Tractable CON

Original  Missing  Conditional sample

Peharz et al., “Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits”, 2020
Generative models are still hard to control
Generate a sentence using "frisbee", "caught" and "dog", following the given order.
Generate a sentence using "frisbee", "caught" and "dog", following the given order.

After a perfect throw, the frisbee glided through the air, and the dog, with incredible agility, caught it mid-flight.
Generate a sentence using "frisbee", "caught" and "dog", following the given order.

After a perfect throw, the frisbee glided through the air, and the dog, with incredible agility, caught it mid-flight.

That's not correct. Generate a sentence using "frisbee", "caught" and "dog". The keywords should appear in the order as specified.

Here's the correct sentence: The dog caught the frisbee in mid-air, showing off its amazing catching skills.
Generate a sentence using "frisbee", "caught" and "dog", following the given order.

After a perfect throw, the frisbee glided through the air, and the dog, with incredible agility, caught it mid-flight.

That's not correct. Generate a sentence using "frisbee", "caught" and "dog". The keywords should appear in the order as specified.

Here's the correct sentence: The dog caught the frisbee in mid-air, showing off its amazing catching skills.

A frisbee is caught by a dog.
A pair of frisbee players are caught in a dog fight.
What do we have?

Prefix: “The weather is”

Constraint α: text contains “winter”

Model only does $p(\text{next-token}|\text{prefix}) =$

<table>
<thead>
<tr>
<th>cold</th>
<th>0.05</th>
</tr>
</thead>
<tbody>
<tr>
<td>warm</td>
<td>0.10</td>
</tr>
</tbody>
</table>
What do we need?

Prefix: “The weather is”

Constraint $\alpha$: text contains “winter”

Generate from $p(\text{next-token}|\text{prefix, } \alpha) = \propto \sum_{\text{text}} p(\text{next-token, text, prefix, } \alpha)$

Marginalization!
Computing $p(\alpha \mid x_{1:t+1})$

For $\alpha$ in conjunctive normal form (CNF):

$$(w_{1,1} \lor \ldots \lor w_{1,d_1}) \land \ldots \land (w_{m,1} \lor \ldots \lor w_{m,d_m})$$

where each $w_{ij}$ is a keyword (i.e. a string of tokens), representing the constraint that $w_{ij}$ appears in the generated text.

e.g., $\alpha = ("swims" \lor "like swimming") \land ("lake" \lor "pool")$
Computing $p(\alpha \mid x_{1:t+1})$

For $\alpha$ in conjunctive normal form (CNF):

$$(w_{1,1} \lor \ldots \lor w_{1,d_1}) \land \ldots \land (w_{m,1} \lor \ldots \lor w_{m,d_m})$$

where each $w_{ij}$ is a keyword (i.e. a string of tokens), representing the constraint that $w_{ij}$ appears in the generated text.

e.g., $\alpha = ("swims" \lor "like swimming") \land ("lake" \lor "pool")$

Efficient algorithm:
For $m$ clauses and sequence length $n$, time-complexity for generation is $O(2^{|m|}n)$ when $p$ is a hidden Markov model (see general probabilistic circuit case later).

Trick: dynamic programming with clever preprocessing and local belief updates

CommonGen: a Challenging Benchmark

Given 3-5 concepts (keywords), our goal is to generate a sentence using all keywords, which can appear in any order and any form of inflections. e.g.,

**Input**: snow drive car

**Reference 1**: A car drives down a snow covered road.

**Reference 2**: Two cars drove through the snow.

\[(w_{1,1} \lor \ldots \lor w_{1,d_1}) \land \ldots \land (w_{m,1} \lor \ldots \lor w_{m,d_m})\]

Each clause represents the inflections for one keyword.
**GeLaTo**

**Overview**

Lexical Constraint $\alpha$: sentence contains keyword "winter"

Constrained Generation: $\Pr(x_{t+1} | \alpha, x_{1:t} = "the weather is")$

| $x_{t+1}$ | $\Pr_{LM}(x_{t+1} | x_{1:t})$ | $\Pr_{TPM}(\alpha | x_{t+1}, x_{1:t})$ |
|-----------|-------------------------------|----------------------------------|
| cold      | 0.05                          | 0.50                             |
| warm      | 0.10                          | 0.01                             |

Pre-trained Language Model

Tractable Probabilistic Model

Minimize KL-divergence

GeLaTo Overview

Constrained Generation: Pr($x_{t+1} \mid \alpha, x_{1:t} = "the weather is")$)

Pre-trained Language Model

Tractable Probabilistic Model

Minimize KL-divergence

<table>
<thead>
<tr>
<th>$x_{t+1}$</th>
<th>$\Pr_{LM}(x_{t+1} \mid x_{1:t})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>cold</td>
<td>0.05</td>
</tr>
<tr>
<td>warm</td>
<td>0.10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_{t+1}$</th>
<th>$\Pr_{TPM}(\alpha \mid x_{t+1}, x_{1:t})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>cold</td>
<td>0.50</td>
</tr>
<tr>
<td>warm</td>
<td>0.01</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_{t+1}$</th>
<th>$p(x_{t+1} \mid \alpha, x_{1:t})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>cold</td>
<td>0.025</td>
</tr>
<tr>
<td>warm</td>
<td>0.001</td>
</tr>
</tbody>
</table>
Step 2: Control $p_{gpt}$ via $p_{hmm}$

**Unsupervised**

Language model is not fine-tuned/prompted to satisfy constraints

By Bayes rule:

$$p_{gpt}(x_{t+1} | x_{1:t}, \alpha) \propto p_{gpt}(\alpha | x_{1:t+1}) \cdot p_{gpt}(x_{t+1} | x_{1:t})$$

Assume $p_{hmm}(\alpha | x_{1:t+1}) \approx p_{gpt}(\alpha | x_{1:t+1})$, we generate from:

$$p(x_{t+1} | x_{1:t}, \alpha) \propto p_{hmm}(\alpha | x_{1:t+1}) \cdot p_{gpt}(x_{t+1} | x_{1:t})$$

<table>
<thead>
<tr>
<th>Method</th>
<th>Generation Quality</th>
<th>Constraint Satisfaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dev</td>
<td>test</td>
</tr>
<tr>
<td>InsNet (Lu et al., 2022a)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>NeuroLogic (Lu et al., 2021)</td>
<td>-  41.9</td>
<td>-</td>
</tr>
<tr>
<td>A*esque (Lu et al., 2022b)</td>
<td>-</td>
<td>44.3</td>
</tr>
<tr>
<td>NADO (Meng et al., 2022)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GeLaTo</td>
<td>44.6</td>
<td>44.1</td>
</tr>
</tbody>
</table>

### Step 2: Control $p_{gpt}$ via $p_{hmm}$

**Supervised**

Language model is fine-tuned to perform constrained generation (e.g. seq2seq)

Empirically $p_{HMM}(\alpha | x_{1:t+1}) \approx p_{gpt}(\alpha | x_{1:t+1})$ does not hold well enough;

we view $p_{HMM}(x_{t+1} | x_{1:t}, \alpha)$ and $p_{gpt}(x_{t+1} | x_{1:t})$ as classifiers trained for the same task with different biases; thus we generate from their weighted geometric mean:

$$p(x_{t+1} | x_{1:t}, \alpha) \propto p_{hmm}(x_{t+1} | x_{1:t}, \alpha)^w \cdot p_{gpt}(x_{t+1} | x_{1:t})^{1-w}$$

| Method                  | ROUGE-L |  | Generation Quality |  | CIDEr |  | SPICE |  | Constraint Satisfaction |  | Coverage |  | Success Rate |
|-------------------------|---------|----------------|-----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| **Supervised**          | dev     | test           | dev             | test           | dev             | test           | dev             | test           | dev             | test           | dev             | test           |
| NeuroLogic (Lu et al., 2021) | -       | 42.8           | -               | 26.7           | -               | 14.7           | -               | 30.5           | -               | 97.7           | -               | 93.9†          |
| A*esque (Lu et al., 2022b) | -       | 43.6           | -               | 28.2           | -               | 15.2           | -               | 30.8           | -               | 97.8           | -               | 97.9†          |
| NADO (Meng et al., 2022)  | 44.4†   | -              | 30.8            | -               | 16.1†           | 32.0†           | -               | 97.1           | -               | 88.8†          | -               |
| GeLaTo                  | 46.0    | 45.6           | 34.1            | 32.9           | 16.7            | 16.8           | 31.3            | 31.9           | 100.0           | 100.0          | 100.0          | 100.0          |

Advantages of GeLaTo:

1. Constraint $\alpha$ is guaranteed to be satisfied: for any next-token $x_{t+1}$ that would make $\alpha$ unsatisfiable, $p(x_{t+1} | x_{1:t}, \alpha) = 0$ for both settings.

2. Training $p_{\text{hmm}}$ does not depend on $\alpha$, which is only imposed at inference (generation) time. Once $p_{\text{hmm}}$ is trained, we can impose whatever $\alpha$.

3. We can impose additional tractable constraints:
   - The keywords are generated following a particular order.
   - (Some) keywords must appear at a particular position.
   - (Some) keywords must not appear in the generated sentence.

Conclusion: you can control an intractable generative model using a tractable probabilistic circuit.
Smoothness + decomposability = tractable MAP

We can also decompose bottom-up a MAP query:

$$\max_{q} p(q \mid e)$$
**Smoothness** + **decomposability** = **tractable MAP**

We *cannot* decompose bottom-up a MAP query:

\[
\max_q p(q \mid e)
\]

since for a sum node we are marginalizing out a latent variable

\[
\max_q \sum_i w_i p_i(q, e) = \max_q \sum z p(q, z, e) \neq \sum z \max_q p(q, z, e)
\]

⇒ **MAP for latent variable models is intractable** (Conaty et al. 2017)
**Determinism**

*aka selectivity*

A sum node is deterministic if the output of only one children is non zero for any input

⇒ e.g. if their distributions have disjoint support

Darwiche and Marquis, “A knowledge compilation map”, 2002
Determinism + decomposability = tractable MAP

Computing maximization with arbitrary evidence $e$  
\[ \implies \text{linear in circuit size!} \]

E.g., suppose we want to compute:
\[
\max_q p(q \mid e)
\]
Determinism + decomposability = tractable MAP

If \( p(q, e) = \sum_i w_i p_i(q, e) = \max_i w_i p_i(q, e) \),
(deterministic sum node):

\[
\max_q p(q, e) = \max_q \sum_i w_i p_i(q, e) = \max_q \max_i w_i p_i(q, e) = \max_i \max_q w_i p_i(q, e)
\]

\( \Rightarrow \) one non-zero child term, thus sum is max
Determinism + decomposability = tractable MAP

If \( p(q, e) = p(q_x, e_x, q_y, e_y) = p(q_x, e_x)p(q_y, e_y) \) (decomposable product node):

\[
\max_q p(q \mid e) = \max_q p(q, e) \\
= \max_{q_x, q_y} p(q_x, e_x, q_y, e_y) \\
= \max_{q_x} p(q_x, e_x) \cdot \max_{q_y} p(q_y, e_y)
\]

\( \Rightarrow \) solving optimization independently
**Determinism** + **decomposability** = **tractable MAP**

Evaluating the circuit twice:

- **bottom-up** and **top-down**

⇒ **still linear in circuit size!**
Evaluating the circuit twice:

bottom-up and top-down \implies still linear in circuit size!

E.g., for $\text{argmax}_{x_1, x_3} p(x_1, x_3 \mid x_2, x_4)$:

1. turn sum into max nodes and distributions into max distributions

2. evaluate $p(x_2, x_4)$ bottom-up

3. retrieve max activations top-down

4. compute MAP states for $X_1$ and $X_3$ at leaves
**Determinism** + **decomposability** = **tractable MAP**

Evaluating the circuit twice: **bottom-up** and **top-down** ⇒ still linear in circuit size!

E.g., for $\text{argmax}_{x_1, x_3} p(x_1, x_3 \mid x_2, x_4)$:

1. turn sum into max nodes and distributions into max distributions
2. evaluate $p(x_2, x_4)$ bottom-up
3. retrieve max activations top-down
4. compute **MAP states** for $X_1$ and $X_3$ at leaves
Determinism + decomposability = tractable MAP

Evaluating the circuit twice:
bottom-up and top-down ⇒ still linear in circuit size!

E.g., for \( \text{argmax}_{x_1, x_3} p(x_1, x_3 \mid x_2, x_4) \):

1. turn sum into max nodes and distributions into max distributions
2. evaluate \( p(x_2, x_4) \) bottom-up
3. retrieve max activations top-down
4. compute MAP states for \( X_1 \) and \( X_3 \) at leaves

![Diagram](image.png)
Determinism + decomposability = tractable MAP

Evaluating the circuit twice:
- **bottom-up** and **top-down** ⇒ still linear in circuit size!

E.g., for argmax_{x_1,x_3} p(x_1, x_3 | x_2, x_4):
1. turn sum into max nodes and distributions into max distributions
2. evaluate p(x_2, x_4) bottom-up
3. retrieve max activations top-down
4. compute **MAP states** for X_1 and X_3 at leaves
Semantic segmentation is MAP over joint pixel and label space
Even approximate MAP for non-deterministic circuits (SPNs) delivers good performances.

Rathke et al., “Locally adaptive probabilistic models for global segmentation of pathological oct scans”, 2017
**How expressive?**

competitive with Flows and VAEs!

---

Dang et al., “Sparse Probabilistic Circuits via Pruning and Growing”, 2022

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Sparse PC (ours)</th>
<th>HCLT</th>
<th>RatSPN</th>
<th>IDF</th>
<th>BitSwap</th>
<th>BB-ANS</th>
<th>McBits</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>1.14</td>
<td>1.20</td>
<td>1.67</td>
<td>1.90</td>
<td>1.27</td>
<td>1.39</td>
<td>1.98</td>
</tr>
<tr>
<td>EMNIST (MNIST)</td>
<td>1.52</td>
<td>1.77</td>
<td>2.56</td>
<td>2.07</td>
<td>1.88</td>
<td>2.04</td>
<td>2.19</td>
</tr>
<tr>
<td>EMNIST (Letters)</td>
<td>1.58</td>
<td>1.80</td>
<td>2.73</td>
<td>1.95</td>
<td>1.84</td>
<td>2.26</td>
<td>3.12</td>
</tr>
<tr>
<td>EMNIST (Balanced)</td>
<td>1.60</td>
<td>1.82</td>
<td>2.78</td>
<td>2.15</td>
<td>1.96</td>
<td>2.23</td>
<td>2.88</td>
</tr>
<tr>
<td>EMNIST (ByClass)</td>
<td>1.54</td>
<td>1.85</td>
<td>2.72</td>
<td>1.98</td>
<td>1.87</td>
<td>2.23</td>
<td>3.14</td>
</tr>
<tr>
<td>FashionMNIST</td>
<td>3.27</td>
<td>3.34</td>
<td>4.29</td>
<td>3.47</td>
<td>3.28</td>
<td>3.66</td>
<td>3.72</td>
</tr>
</tbody>
</table>
### How scalable?


<table>
<thead>
<tr>
<th>Dataset</th>
<th>TPMs</th>
<th>DGMs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LVD (ours)</td>
<td>HCLT</td>
</tr>
<tr>
<td>ImageNet32</td>
<td>4.39±0.01</td>
<td>4.82</td>
</tr>
<tr>
<td>ImageNet64</td>
<td>4.12±0.00</td>
<td>4.67</td>
</tr>
<tr>
<td>CIFAR</td>
<td>4.38±0.02</td>
<td>4.61</td>
</tr>
</tbody>
</table>

![ImageNet64 graph showing scalability](image)
Questions answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Questions answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Logical Circuits
Tractability to other semi-rings

Tractable probabilistic inference exploits **efficient summation for decomposable functions** in the probability commutative semiring:

\[(\mathbb{R}, +, \times, 0, 1)\]

analogously efficient computations can be done in other semi-rings:

\[(\mathcal{S}, \oplus, \otimes, 0_\oplus, 1_\otimes)\]

\[\Rightarrow \text{ Algebraic model counting (Kimmig et al. 2017), Semi-ring programming (Belle et al. 2016)}\]

Historically, **very well studied for boolean functions**:

\[(\mathbb{B} = \{0, 1\}, \lor, \land, 0, 1)\]

\[\Rightarrow \text{ logical circuits!}\]
Logical circuits are compact representations for boolean functions...
Logical circuits

...and like probabilitistic circuits, one can define **structural properties**: (structured) decomposability, smoothness, determinism allowing for tractable computations.

---

**Decomposability**

**Determinism**

**Smoothness**

---

*Darwiche and Marquis, “A knowledge compilation map”, 2002*
Logical circuits

a knowledge compilation map

...inducing a hierarchy of tractable logical circuit families

Darwiche and Marquis, “A knowledge compilation map”, 2002
Knowledge Compilation

encoding

(A and (not B))
or(C and (not D))
or ((not C) and D)
...

Compiler

NNF Circuit

\[ \neg A \quad B \quad \neg B \quad A \quad \neg C \quad D \quad \neg D \quad \neg C \]

MAJ-MAJ-SAT
E-MAJ-SAT
MAJ-SAT
SAT

Answer in
Linear Time
NNF Circuits

\[ P \lor L \]
\[ A \Rightarrow P \]
\[ K \Rightarrow (P \lor L) \]
Decomposability (DNNF)

Darwiche, JACM 2001

SAT in linear time
Determinism (d-DNNF)

Darwiche, JANCL 2000

MAJ-SAT in linear time

Input: $L, K, P, A$
Decomposability + determinism = tractable (W)MC

**Model counting problem:** given a Boolean formula $\Delta$, compute the number of satisfying assignments.

**Weighted model counting (WMC):**

$$WMC(\Delta, w) = \sum_{x \models \Delta} \prod_{l \in x} w(l)$$

$\Rightarrow$ linear in circuit size!
Decomposability + determinism = tractable (W)MC

To compute $WMC(\Delta, w)$:

- Turn OR gates to sum nodes and AND gates to product nodes
- Replace each literal $l$ with its weight $w(l)$
- Bottom-up evaluation
Decomposability + determinism = tractable (W)MC

To compute $WMC(\Delta, w)$:

- Turn OR gates to sum nodes and AND gates to product nodes
- Replace each literal $l$ with its weight $w(l)$
- Bottom-up evaluation
Probabilistic inference by WMC
connection to probabilistic circuits through WMC

1. Encode probabilistic model as WMC formula \((\Delta, w)\)
2. Compile \(\Delta\) into a logical circuit (e.g. d-DNNF, OBDD, SDD, etc.)
3. Tractable MAR/CON by tractable WMC on circuit
4. Answer complex queries tractably by enforcing more structural properties!

Probabilistic inference by WMC

connection to probabilistic circuits through WMC

Resulting compiled WMC circuit \textit{equivalent to probabilistic circuit} \Rightarrow \textit{parameter variables} \rightarrow \textit{edge parameters}

Compiled circuit of WMC encoding

Equivalent probabilistic circuit
Questions answered today

1. *What are probabilistic queries? Are current models tractable?* (Guy)
2. *What are probabilistic circuits and why are they tractable?* (Guy)
3. *What is the connection to logical circuit languages?* (YooJung)
4. *How do I compile my favorite model into a circuit?* (YooJung)
5. *How are circuit size and tractability related?* (YooJung)
6. *What’s the most impressive query we can efficiently compute?* (YooJung)
7. *Are all tractable distributions probabilistic circuits?* (Guy)
8. *How to learn probabilistic circuits from data?* (Guy)
Questions answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. **How do I compile my favorite model into a circuit?** (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
From tree BN to circuits via compilation
From tree BN to circuits via compilation

Bottom-up *compilation*: starting from leaves...
From tree BN to circuits via compilation

...compile a leaf CPT

\[ p(A|C = 0) \]

\[ \begin{align*}
A = 0 & \quad p(A|C = 0) = 0.3 \\
A = 1 & \quad p(A|C = 0) = 0.7
\end{align*} \]
From tree BN to circuits
via compilation

...compile a leaf CPT

\[ p(A|C = 1) \]

\[ \begin{array}{c}
\text{A = 0} \\
\text{A = 1}
\end{array} \]

\[ \begin{array}{c}
\text{.6} \\
\text{.4}
\end{array} \]
From tree BN to circuits
via compilation

...compile a leaf CPT...for all leaves...

\[ p(A|C) \]
\[ p(B|C) \]
From tree BN to circuits via compilation

...and recurse over parents...

\[ p(C|D = 0) \]

\[ \times \]

\[ .2 \times .8 \]

\[ \times \]

\[ A = 0 \quad A = 1 \quad B = 0 \quad B = 1 \]

\[ C = 1 \]
From tree BN to circuits via compilation

...while reusing previously compiled nodes!
From tree BN to circuits via compilation

\[ A = 0 \quad A = 1 \quad B = 0 \quad B = 1 \]
\[ C = 0 \quad C = 1 \]
\[ D = 0 \quad D = 1 \]

\[ p(D) \]
\[ \times \]

\[ A \]
\[ B \]
\[ C \]
\[ D \]
Hidden Markov Models

as computational graphs
<table>
<thead>
<tr>
<th>Line 1</th>
<th>Lines 1-6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line 2</td>
<td>Lines 2-6</td>
</tr>
<tr>
<td>Line 3</td>
<td>Line 3</td>
</tr>
<tr>
<td>Line 4</td>
<td>Line 4</td>
</tr>
<tr>
<td>Line 5</td>
<td>Line 5</td>
</tr>
<tr>
<td>Line 6</td>
<td>Line 6</td>
</tr>
</tbody>
</table>

| 1     | x = flip(θ₁);   |
| 2     | if(x) {         |
| 3     | y = flip(θ₂)    |
| 4     | } else {        |
| 5     | y = x           |
| 6     | }               |

Chavira et al., “Compiling relational Bayesian networks for exact inference”, 2006
Holtzen et al., “Symbolic Exact Inference for Discrete Probabilistic Programs”, 2019
Vlasselaer et al., “Exploiting Local and Repeated Structure in Dynamic Bayesian Networks”, 2016
Decision Diagrams

- FBDDs (Free binary decision diagrams; read-once)
- OBDDs (Ordered BDDs)
- SDDs (Sentential decision diagrams)

BDD as circuit

Darwiche and Marquis, “A knowledge compilation map”, 2002
Structured Decomposability

Pipatsrisawat & Darwiche, AAAI 2008
Structured Decomposability

Pipatsrisawat & Darwiche, AAAI 2008
Structured Decomposability

Pipatsrisawat & Darwiche, AAAI 2008
Partitioned Determinism (SDDs)

Input: \( L, K, P, A \)

Darwiche, IJCAI 2011
Partitioned Determinism (SDDs)

Input: $L$, $K$, $P$, $A$

Darwiche, IJCAI 2011
**Decision Diagrams**

- FBDDs (Free binary decision diagrams; \textit{read-once})
- OBDDs (Ordered BDDs)
- SDDs (Sentential decision diagrams)

\[ (A \land B) \lor (C \land D) \]

What is the probability of having a traffic jam on my route to campus?
**Probability of logical events**

$q_8$: What is the probability of having a traffic jam on my route to campus?

$q_8(m) = p_m(\bigvee_{i \in \text{route}} \text{JamStr}_i)$

$\Rightarrow$ marginals + logical events
**Smoothness** + **structured decomp.** = **tractable PR**

Computing $p(\alpha)$: the probability of arbitrary logical formula

Multilinear in circuit sizes if the logical circuit:
- is smooth, structured decomposable, deterministic
- shares the same vtrees
Smoothness + structured decomp. = tractable PR

If \( p(x) = \sum_i w_i p_i(x) \), \( \alpha = \bigvee_j \alpha_j \),
(smooth \( p \))
(smooth + deterministic \( \alpha \)):

\[
p(\alpha) = \sum_i w_i p_i \left( \bigvee_j \alpha_j \right) = \sum_i w_i \sum_j p_i(\alpha_j)
\]

\[\Rightarrow \text{probabilities are “pushed down” to children}\]
If \( p(x, y) = p(x)p(y) \), \( \alpha = \beta \land \gamma \),

*structured decomposability*:

\[
p(\alpha) = p(\beta \land \gamma) \cdot p(\beta \land \gamma) = p(\beta) \cdot p(\gamma)
\]

\[\implies\] probabilities decompose into simpler ones
Smoothness + structured decomp. = tractable PR

To compute $p(\alpha)$:
- compute the probability for each pair of probabilistic and logical circuit nodes for the same vtree node
  $\implies$ cache the values!
- feedforward evaluation (bottom-up)
Smoothness + structured decomp. = tractable PR

To compute $p(\alpha)$:

- compute the probability for each pair of probabilistic and logical circuit nodes for the same vtree node
  $\implies$ cache the values!

- feedforward evaluation (bottom-up)
structured decomposability = tractable...

- **Symmetric** and **group queries** (exactly-$k$, odd-number, etc.) (Bekker et al. 2015)

For the “right” vtree

- Marginal MAP (Oztok et al. 2016)
- Probability of logical circuit event in probabilistic circuit (Choi et al. 2015b)
- **Multiply** two probabilistic circuits (Shen et al. 2016)
- **KL Divergence** between probabilistic circuits (Liang et al. 2017)
- **Same-decision probability** (Oztok et al. 2016)
- **Expected same-decision probability** (Choi et al. 2017)
- **Expected classifier agreement** (Choi et al. 2018)
- **Expected predictions** (Khosravi et al. 2019b)
Questions answered today

1. *What are probabilistic queries? Are current models tractable?* (Guy)
2. *What are probabilistic circuits and why are they tractable?* (Guy)
3. *What is the connection to logical circuit languages?* (YooJung)
4. *How do I compile my favorite model into a circuit?* (YooJung)
5. *How are circuit size and tractability related?* (YooJung)
6. *What’s the most impressive query we can efficiently compute?* (YooJung)
7. *Are all tractable distributions probabilistic circuits?* (Guy)
8. *How to learn probabilistic circuits from data?* (Guy)
Questions answered today

1. *What are probabilistic queries? Are current models tractable?* (Guy)
2. *What are probabilistic circuits and why are they tractable?* (Guy)
3. *What is the connection to logical circuit languages?* (YooJung)
4. *How do I compile my favorite model into a circuit?* (YooJung)
5. **How are circuit size and tractability related?** (YooJung)
6. *What’s the most impressive query we can efficiently compute?* (YooJung)
7. *Are all tractable distributions probabilistic circuits?* (Guy)
8. *How to learn probabilistic circuits from data?* (Guy)
Succinctness of circuits

Expressive efficiency

Tractability is defined with respect to the size of the model.

How do structural constraints affect **expressive efficiency** (**succinctness**) of probabilistic/logical circuits?
Succinctness of circuits

Expressive efficiency

A family of circuits $\mathcal{M}_1$ is at least as succinct as $\mathcal{M}_2$ iff for every $m_2 \in \mathcal{M}_2$, there exists $m_1 \in \mathcal{M}_1$ that represents the same function and $|m_1| \leq |\text{poly}(m_2)|$.

$\Rightarrow$ denoted $\mathcal{M}_1 \leq \mathcal{M}_2$

$\Rightarrow$ strictly more succinct ($\mathcal{M}_1 < \mathcal{M}_2$) iff $\mathcal{M}_1 \leq \mathcal{M}_2$ and $\mathcal{M}_1 \not\geq \mathcal{M}_2$
Succinctness of circuits

Expressive efficiency

Strict succinctness ordering: DNNF < d-DNNF < FBDD < OBDD

Darwiche and Marquis, “A knowledge compilation map”, 2002
Succinctness of circuits

Expressive efficiency

Strict succinctness ordering: $\text{DNNF} < \text{d-DNNF} < \text{FBDD} < \text{OBDD}$

- $\text{d-DNNF} \not\leq \text{DNNF}$ unless the polynomial hierarchy collapses (Darwiche et al. 2002a).

- The Sauerhoff function has DNNF of size $O(n^2)$ but d-DNNF of size $2^{\Omega(n)}$ (Bova et al. 2016).
Succinctness of circuits

Expressive efficiency

Strict succinctness ordering: DNNF < d-DNNF < FBDD < OBDD

- d-DNNF $\not\leq$ DNNF unless the polynomial hierarchy collapses (Darwiche et al. 2002a).
- The Sauerhoff function has DNNF of size $O\left(n^2\right)$ but d-DNNF of size $2^{\Omega(n)}$ (Bova et al. 2016).

$\implies$ Unconditional exponential separation for d-DNNF $\not\leq$ DNNF

$\implies$ Using a connection between circuits and communication complexity
**Succinctness of circuits**

**Expressive efficiency**

**SDD \textless OBDD**: SDDs are strictly more succinct than OBDDs

- SDD \textless= OBDD: OBDDs are SDDs with right-linear vtrees
- SDD \textgreater= OBDD: The *hidden weighted bit function* has SDD of size $O(n^3)$ but OBDD of size $2^{\Omega(n)}$.

---

Bova, “SDDs are exponentially more succinct than OBDDs”, 2016
Möbius Über Alles

PTIME

Poly-size FBDD, dec-DNNF

Poly-size OBDD, SDD = inversion-free

Read Once

∀FO_un, ∃FO_un

Non-hierarchical

hierarchical
How precise is the characterization of tractable circuits by structural properties?
Smoothness + decomposability = tractable MAR

Recall: Smoothness and decomposability allow marginal inference by feedforward (sum-product) evaluation.
Recall: Smoothness and decomposability allow marginal inference by feedforward (sum-product) evaluation.

⇒ Are these properties necessary?
Smoothness + decomposability = tractable MAR

Recall: Smoothness and decomposability allow marginal inference by feedforward (sum-product) evaluation.

⇒ Are these properties necessary?
⇒ Yes! Otherwise, integrals do not decompose.
Determinism + decomposability = tractable MAP

Recall: Determinism and decomposability allow MAP inference by feedforward (max-product) evaluation.
**Determinism** + **decomposability** = **tractable MAP**

Recall: Determinism and decomposability allow MAP inference by feedforward (max-product) evaluation.

⇒ However, decomposability is not necessary!
**Determinism** + **decomposability** = **tractable MAP**

Recall: Determinism and decomposability allow MAP inference by feedforward (max-product) evaluation.

⇒ However, decomposability is not necessary!
⇒ A weaker condition, consistency, suffices.
A product node is consistent if any variable shared between its children appears in a single leaf node.

$\Rightarrow$ decomposability implies consistency

**Consistent circuit**

$w_1 w_2 w_3 w_4$

$X_1 X_2 X_3$

- $X_1 \land X_2$\hspace{1cm} $X_2 \land X_3$

$w_1 w_2 w_3 w_4$

$X_1 X_2 X_3$

**Inconsistent circuit**

$w_1 w_2 w_3 w_4$

$X_1 \land X_2 \leq \theta$\hspace{1cm} $X_2 > \theta X_3$

$w_1 w_2 w_3 w_4$

$X_1 X_2 X_3$
Determinism + consistency = tractable MAP
Determinism + consistency = tractable MAP

If \( \max_{q_{\text{shared}}} p(q, e) = \max_{q_{\text{shared}}} p(q_x, e_x) \cdot \max_{q_{\text{shared}}} p(q_y, e_y) \) (consistent):

\[
\max_q p(q, e) = \max_{q_x, q_y} p(q_x, e_x, q_y, e_y) = \max_{q_x} p(q_x, e_x) \cdot \max_{q_y} p(q_y, e_y)
\]

\( \Rightarrow \) solving optimization independently
Expressive efficiency of circuits

Are smooth & decomposable circuits as succinct as deterministic & consistent ones, or vice versa?
Expressive efficiency of circuits

Smooth & decomposable circuits strictly more succinct than deterministic & decomposable ones.

Smooth & consistent circuits are equally succinct as smooth & decomposable ones.

\[\text{det. & Decomp.}\]
Expressive efficiency of circuits

- Smooth & decomposable circuits strictly more succinct than deterministic & decomposable ones
- Smooth & consistent circuits are equally succinct as smooth & decomposable ones

(Peharz et al. 2015)*

(Darwiche et al. 2002b)

: strictly more succinct

: equally succinct
Expressive efficiency of circuits

Smooth & decomposable circuits strictly more succinct than deterministic & decomposable ones.

Smooth & consistent circuits are equally succinct as smooth & decomposable ones.

- Strictly more succinct
- Equally succinct
Consider following circuit over Boolean variables:
\[ \prod_i (Y_i \cdot Z_{i1} + (\neg Y_i) \cdot Z_{i2}), \quad Z_{ij} \in \mathbf{X} \]
- Size linear in the number of variables
- Deterministic and consistent
- Marginal (with no evidence) is the solution to \#P-hard SAT’ problem (Valiant 1979) ⇒ no tractable circuit for marginals!
Expressive efficiency of circuits

Consider the following circuit over Boolean variables:

$$\prod_i (Y_i \cdot Z_{i1} + (\neg Y_i) \cdot Z_{i2}), \quad Z_{ij} \in X$$

- Size linear in the number of variables
- Deterministic and consistent
- Marginal (with no evidence) is the solution to \#P-hard SAT$'$ problem (Valiant 1979) \Rightarrow no tractable circuit for marginals!

\(\rightarrow\) : strictly more succinct
\(-\rightarrow\) : equally succinct
Consider the marginal distribution $p(X)$ from a naive Bayes distribution $p(X, C)$:

- Linear-size smooth and decomposable circuit
- MAP of $p(X)$ solves marginal MAP of $p(X, C)$ which is NP-hard (de Campos 2011)
  ⇒ no tractable circuit for MAP!
Consider the marginal distribution $p(X)$ from a naive Bayes distribution $p(X, C)$:

- Linear-size smooth and decomposable circuit
- MAP of $p(X)$ solves marginal MAP of $p(X, C)$ which is NP-hard *(de Campos 2011)*

$\Rightarrow$ **no tractable circuit for MAP!**
Expressive efficiency of circuits

Neither smooth & decomposable nor deterministic & consistent circuits are more succinct than the other!

⇒ Choose tractable circuit family based on your query

More theoretical questions remaining

⇒ “Complete the map”

: strictly more succinct
— : equally succinct
Neither smooth & decomposable nor deterministic & consistent circuits are more succinct than the other!

⇒ Choose tractable circuit family based on your query

More theoretical questions remaining
⇒ “Complete the map”
Expressive efficiency of circuits

Succinctness map for *monotone* circuits

⇒ (s)mooth, (d)eterministic, (D)ecomposable, (w)eak (D)ecomposable (i.e. consistent)

Colnet and Mengel, “A Compilation of Succinctness Results for Arithmetic Circuits”, 2021
Expressive efficiency of circuits

Succinctness map for **monotone** circuits

\[
\begin{align*}
\text{sdwD-AC}_m & \rightarrow \text{sdD-AC}_m \\
\text{swD-AC}_m & \rightarrow \text{sD-AC}_m \\
\text{dwD-AC}_m & \rightarrow \text{dD-AC}_m \\
\text{wD-AC}_m & \rightarrow \text{D-AC}_m
\end{align*}
\]

Succinctness map for **positive** circuits

(no-negative output, but weights may be negative)

\[
\begin{align*}
\text{sdwD-AC}_p & \rightarrow \text{sdD-AC}_p \\
\text{swD-AC}_p & \rightarrow \text{sD-AC}_p \\
\text{dwD-AC}_p & \rightarrow \text{dD-AC}_p \\
\text{wD-AC}_p & \rightarrow \text{D-AC}_p
\end{align*}
\]

⇒ (s)mooth, (d)eterministic, (D)ecomposable, (w)eak (D)ecomposable (i.e. consistent)

Colnet and Mengel, “A Compilation of Succinctness Results for Arithmetic Circuits”, 2021
Questions answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Questions answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Goal

Given a class of queries, can we systematically find a class of probabilistic circuits that is tractable for it?
A language for queries

Integral expressions that can be formed by composing these operators

\[ +, \times, \text{pow}, \log, \exp \text{ and } / \]

\[ \Rightarrow \text{ many divergences and information-theoretic queries} \]
A language for queries

Integral expressions that can be formed by composing these operators

\(+, \times, \text{pow}, \log, \exp\) and \(/\)

\(\Rightarrow\) many divergences and information-theoretic queries

Represented as **higher-order computational graphs**—pipelines—operating over circuits!

\(\Rightarrow\) re-using intermediate transformations across queries
\[ \text{KLD}(p \parallel q) = \int_{\text{val}(X)} p(x) \times \log \left( \frac{p(x)}{q(x)} \right) \, dX \]
\[ \text{KLD}(p \parallel q) = \int_{\text{val}(\mathbf{x})} p(\mathbf{x}) \times \log \left( \frac{p(\mathbf{x})}{q(\mathbf{x})} \right) \, d\mathbf{X} \]
\[
\text{KLD}(p \parallel q) = \int_{\text{val}(x)} p(x) \times \log \left( \frac{p(x)}{q(x)} \right) \, dX
\]
\[ \text{KLD}(p \parallel q) = \int_{\text{val}(x)} p(x) \times \log \left( \frac{p(x)}{q(x)} \right) \, dX \]
$$\text{XENT}(p \parallel q) = \int p(x) \times \log q(x) \, dX$$
\[ \mathbb{E}_{x^m \sim p(x^m | x^o)} \left[ q^\alpha(x^m, x^o) \right] \]
Compatibility

Two circuits are *compatible* if they have the same *hierarchical scope partitioning* ⇒ generalizes “structured decomposability with same vtree”

**compatible circuits**

**non-compatible circuits**
Tractable operators

smooth, decomposable compatible
Tractable operators

\[
\begin{align*}
\llbracket X < \gamma \rrbracket & \land \llbracket Y \geq \delta \rrbracket \\
\llbracket X \geq \gamma \rrbracket & \land \llbracket Y < \delta \rrbracket \\
\times & \\
p_1 & \land \theta_1 \\
p_2 & \land \theta_2
\end{align*}
\]

smooth, decomposable
deterministic

\[
\begin{align*}
\llbracket X < \gamma \rrbracket & \land \llbracket Y \geq \delta \rrbracket \\
\llbracket X \geq \gamma \rrbracket & \land \llbracket Y < \delta \rrbracket \\
\times & \\
p_1 & \land \theta_1 \\
p_2 & \land \theta_2
\end{align*}
\]

smooth, decomposable
Building an atlas of composable \textit{tractable atomic operations}
To perform tractable integration we need $s$ to be \textit{smooth and decomposable}...
hence we need \( p \) and \( r \) to be smooth, decomposable and \textit{compatible}...
therefore \( q \) must be smooth, decomposable and \textit{deterministic}...
we can compute $\text{XENT}$ tractably if $p$ and $q$ are smooth, decomposable, compatible and $q$ is deterministic...
<table>
<thead>
<tr>
<th>Query</th>
<th>Tract. Conditions</th>
<th>Hardness</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CROSS ENTROPY</strong></td>
<td>Cmp, q Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td><strong>SHANNON ENTROPY</strong></td>
<td>Sm, Dec, Det</td>
<td>coNP-hard w/o Det</td>
</tr>
<tr>
<td><strong>RéNYI ENTROPY</strong></td>
<td>SD</td>
<td>#P-hard w/o SD</td>
</tr>
<tr>
<td><strong>MUTUAL INFORMATION</strong></td>
<td>Sm, SD, Det*</td>
<td>coNP-hard w/o SD</td>
</tr>
<tr>
<td><strong>KULLBACK-LEIBLER DIV.</strong></td>
<td>Cmp, Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td><strong>RéNYI’S ALPHA DIV.</strong></td>
<td>Cmp, q Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td><strong>ITAKURA-SAITO DIV.</strong></td>
<td>Cmp, Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td><strong>CAUCHY-SCHWARZ DIV.</strong></td>
<td>Cmp</td>
<td>#P-hard w/o Cmp</td>
</tr>
<tr>
<td><strong>SQUARED LOSS</strong></td>
<td>Cmp</td>
<td>#P-hard w/o Cmp</td>
</tr>
</tbody>
</table>

*compositionally* derive the tractability of many more queries

<table>
<thead>
<tr>
<th>Query</th>
<th>Tract. Conditions</th>
<th>Hardness</th>
</tr>
</thead>
<tbody>
<tr>
<td>CROSS ENTROPY</td>
<td>$- \int p(x) \log q(x) , dx$</td>
<td>Cmp, q Det</td>
</tr>
<tr>
<td>SHANNON ENTROPY</td>
<td>$-\sum p(x) \log p(x)$</td>
<td>Sm, Dec, Det</td>
</tr>
<tr>
<td>RÉNYI ENTROPY</td>
<td>$(1-\alpha)\log \int p^\alpha(x) , dx$, $\alpha \in \mathbb{N}$</td>
<td>SD</td>
</tr>
<tr>
<td></td>
<td>$(1-\alpha)\log \int p^\alpha(x) , dx$, $\alpha \in \mathbb{R}^+$</td>
<td>Sm, Dec, Det</td>
</tr>
<tr>
<td>MUTUAL INFORMATION</td>
<td>$\int p(x, y) \log(p(x, y)/(p(x)p(y))) , dx$</td>
<td>Sm, SD, Det*</td>
</tr>
<tr>
<td>KULLBACK-LEIBLER DIV.</td>
<td>$\int p(x) \log(p(x)/q(x)) , dx$</td>
<td>Cmp, Det</td>
</tr>
<tr>
<td>RÉNYI’S ALPHA DIV.</td>
<td>$(1-\alpha)\log \int p^\alpha(x)q^{1-\alpha}(x) , dx$, $\alpha \in \mathbb{N}$</td>
<td>Cmp, q Det</td>
</tr>
<tr>
<td></td>
<td>$(1-\alpha)\log \int p^\alpha(x)q^{1-\alpha}(x) , dx$, $\alpha \in \mathbb{R}$</td>
<td>Cmp, Det</td>
</tr>
<tr>
<td>ITAKURA-SAITO DIV.</td>
<td>$\int [p(x)/q(x) - \log(p(x)/q(x))] - 1 , dx$</td>
<td>Cmp, Det</td>
</tr>
<tr>
<td>CAUCHY-SCHWARZ DIV.</td>
<td>$-\log \sqrt{\int p^2(x) , dx} \sqrt{\int q^2(x) , dx}$</td>
<td>Cmp</td>
</tr>
<tr>
<td>SQUARED LOSS</td>
<td>$\int (p(x) - q(x))^2 , dx$</td>
<td>Cmp</td>
</tr>
</tbody>
</table>

*prove hardness* when some input properties are not satisfied

Composable tractable sub-routines

```julia
function kld(p, q)
    r = quotient(p, q)
    s = log(r)
    t = product(p, s)
    return integrate(t)
end

function xent(p, q)
    r = log(q)
    s = product(p, r)
    return -integrate(s)
end

function ent(p)
    q = log(p)
    r = product(p, q)
    return -integrate(s)
end

function alphadiv(p, q, alpha=1.5)
    r = real_pow(p, alpha)
    s = real_pow(q, 1.0-alpha)
    t = product(r, s)
    return log(integrate(t)) / (1.0-alpha)
end
```

Efficient inference algorithms in a couple lines of Julia code!²

²https://github.com/UCLA-StarAI/circuit-ops-atlas
Next up...

1. Learning and reasoning with symbolic constraints

2. Expected predictions: handling missing values, fairness

3. Exact inference of causal effects

⇒ using tractable operators

smooth, decomposable compatible
Symbolic constraints

“How can neural nets reason and learn with symbolic constraints reliably and efficiently?”
When?

Ground Truth

e.g. *predict shortest path in a map*
When?

given $x$  // e.g. a tile map

Ground Truth

structured output prediction (SOP) tasks

Vlastelica et al., “Differentiation of blackbox combinatorial solvers”, 2020
When?  

given \( x \)  // e.g. a tile map

find \( y^* = \text{argmax}_y p_\theta(y \mid x) \)  // e.g. a configurations of edges in a grid

\[ \text{structured output prediction (SOP) tasks} \]

Vlastelica et al., “Differentiation of blackbox combinatorial solvers”, 2020
When?

```
given x  // e.g. a tile map
find y* = arg\max_y p_\theta(y | x)  // e.g. a configurations of edges in a grid
s.t. y \models K  // e.g., that form a valid path
```

*structured output prediction (SOP) tasks*

Vlastelica et al., “Differentiation of blackbox combinatorial solvers”, 2020
When?

Given $x$ // e.g. a tile map

Find $y^* = \arg\max_y p_\theta(y \mid x)$ // e.g. a configurations of edges in a grid

s.t. $y \models K$ // e.g., that form a valid path

// for a $12 \times 12$ grid, $2^{144}$ states but only $10^{10}$ valid ones!

**structured output prediction (SOP) tasks**

Vlastelica et al., “Differentiation of blackbox combinatorial solvers”, 2020
When?

given \( x \)  // e.g. a feature map

find \( y^* = \arg\max_y p_\theta(y \mid x) \)  // e.g. labels of classes

s.t. \( y \models K \)  // e.g., constraints over superclasses

\[ K : (Y_{\text{cat}} \Rightarrow Y_{\text{animal}}) \land (Y_{\text{dog}} \Rightarrow Y_{\text{animal}}) \]

hierarchical multi-label classification

Giunchiglia and Lukasiewicz, “Coherent hierarchical multi-label classification networks”, 2020
neural nets struggle to satisfy domain constraints!
How?

*take an unreliable neural network architecture...*
How?

......and replace the last layer with a semantic probabilistic layer
$q_\Theta(y \mid g(z))$ is an expressive distribution over labels

$\mathcal{c}_K(x, y)$ encodes the constraint $\mathbb{1}\{x, y \models K\}$

\[
p(y \mid x) = q_\Theta(y \mid g(z)) \cdot c_K(x, y)/\mathcal{Z}(x)
\]
\[
\mathcal{Z}(x) = \sum_y q_\Theta(y \mid x) \cdot c_K(x, y)
\]

a conditional circuit $q(y; \Theta = g(z))$
and a logical circuit $c(y, x)$ encoding $K$
Tractable products

\[
X_3 \quad X_2 \quad X_1
\]

\[
X_2 \quad + \quad X_3
\]

\[
X_1 \quad + \quad X_3
\]

\[
X_1 \quad + \quad + \quad X_3
\]

\[
X_1 \quad + \quad + \quad + \quad X_3
\]

smooth, decomposable compatible

exactly compute \( \mathcal{Z} \) in time \( O(||q||c||) \)
SPL recipe

\[ K : (Y_1 = 1 \implies Y_3 = 1) \]
\[ \land (Y_2 = 1 \implies Y_3 = 1) \]

1) Take any logical constraint
**SPL recipe**

\[ K : (Y_1 = 1 \implies Y_3 = 1) \]
\[ \wedge (Y_2 = 1 \implies Y_3 = 1) \]

**1) Take any logical constraint**

**2) Compile it into a constraint circuit**
**SPL recipe**

K : \( (Y_1 = 1 \implies Y_3 = 1) \)
\( \land (Y_2 = 1 \implies Y_3 = 1) \)

1) Take any logical constraint  
2) Compile it into a constraint circuit  
3) Multiply it by a circuit distribution
SPL recipe

\[ K : (Y_1 = 1 \implies Y_3 = 1) \land (Y_2 = 1 \implies Y_3 = 1) \]

1) Take any logical constraint

2) Compile it into a constraint circuit

3) Multiply it by a circuit distribution

4) train end-to-end by sgd!
Guaranteeing consistency

Ground Truth

FIL

$\mathcal{L}_{SL}$

SPL

cost: 39.31
cost: $\infty$
cost: $\infty$
cost: 45.09

cost: 57.31
cost: $\infty$
cost: $\infty$
cost: 58.09
Expected predictions

Reasoning about the output of a classifier or regressor $f$ given a distribution $p$ over the input features

$$\mathbb{E}_p[f] = \int_{\text{val}(X)} p(x) \times f(x) \, dX$$
Handling missing values at test time

Given a partial observation \( \mathbf{x}^o \), what is the expected output from \( f \)?

\[
\mathbb{E}_{\mathbf{x}^m \sim p(\mathbf{x}^m | \mathbf{x}^o)} \left[ f(\mathbf{x}^m, \mathbf{x}^o) \right]
\]

using ProbabilisticCircuits
pc = load_prob_circuit(zoo_psdd_file("insurance.psdd"));
rc = load_logistic_circuit(zoo_lc_file("insurance.circuit"), 1);

q: Is the predictive model biased by gender?

groups = make_observations([["male"], ["female"]])
exps, _ = Expectation(pc, rc, groups);
println("Female : \\
(exps[2])");
println("Male : \\
(exps[1])");
println("Diff : \\
(exps[2] - exps[1])");
Female : $ 14170.125469335406
Male : $ 13196.548926381849
Diff : $ 973.5765429535568

https://github.com/Juice-jl/
Causal Inference

Given subsets $A, Y \subseteq X$, interested in causal effect $p(Y|do(A))$.
Causal Inference

Given subsets $A, Y \subseteq X$, interested in causal effect $p(Y|do(A))$.
In general, $p(Y|do(A)) \neq p(Y|A)$ (correlation is not causation).
Causal Inference

Given subsets $A, Y \subseteq X$, interested in causal effect $p(Y|do(A))$.
In general, $p(Y|do(A)) \neq p(Y|A)$ (correlation is not causation).

- Specify (qualitative) assumptions on the system using a causal diagram $G$ (here $A, Y, Z, K \subseteq X$) :

(a) Backdoor

(b) Napkin

\[
\begin{align*}
&Z \\
&\downarrow \\
&A \quad \rightarrow \quad Y \\
&\quad \\
&K \rightarrow Z \rightarrow A \rightarrow Y
\end{align*}
\]

\[
\begin{align*}
\sum_Z p(Z)p(Y|A, Z) \\
\sum_K p(A,Y|K,Z)p(K)\sum_K p(A|K,Z)p(K)
\end{align*}
\]
Causal Inference

Given subsets $A, Y \subseteq X$, interested in causal effect $p(Y|do(A))$.
In general, $p(Y|do(A)) \neq p(Y|A)$ (correlation is not causation).

- Specify (qualitative) assumptions on the system using a causal diagram $G$ (here $A, Y, Z, K \subseteq X$):

  - Backdoor
    
  - Napkin

- Given causal diagram $G$, can derive expressions for causal effect $p(Y|A)$ using *do-calculus* (Pearl 1995).

\[
\sum_Z p(Z)p(Y|A, Z) \quad \text{(a) Backdoor} \quad \frac{\sum_K p(A,Y|K,Z)p(K)}{\sum_K p(A|K,Z)p(K)} \quad \text{(b) Napkin}
\]
Consider the backdoor query, for fixed values of the treatment $a$ and outcome $y$:

$$p(y|do(a)) := \sum_Z p(Z) \times p(y|a, Z).$$
Consider the backdoor query, for fixed values of the treatment $a$ and outcome $y$:

$$p(y|do(a)) := \sum_Z p(Z) \times p(y|a,Z)$$

Theorem (Wang & Kwiatkowska 2023)

*If $p$ is given as a structured decomposable and deterministic circuit, then the backdoor query is \#P-hard to compute.*
Applying the Atlas of Tractable Operations

Break down do-calculus query into compositions of basic operations, such as marginalization, products, and powers:

(a) Pipeline for \( \text{COND}(\cdot, W) \):

\[
p(V) = MARG(\cdot; V \setminus W) \cdot \text{POW}(\cdot; -1) \cdot \text{PROD}(\cdot, \cdot) \cdot p(V|W)
\]

(b) Pipeline for entire backdoor query

Problem: Cannot guarantee that input to \( \text{POW} \) is deterministic, even if \( p(X) \) is deterministic.
Applying the Atlas of Tractable Operations

Break down do-calculus query into compositions of basic operations, such as marginalization, products, and powers:

(a) Pipeline for $\text{COND}(\cdot, \mathbf{W})$

(b) Pipeline for entire backdoor query

Problem: Cannot guarantee that input to $\text{POW}$ is deterministic, even if $p(\mathbf{X})$ is deterministic.
Applying the Atlas of Tractable Operations

Break down do-calculus query into compositions of basic operations, such as marginalization, products, and powers:

\[ p(V) \]

\[ \text{MARG}(\cdot; V \setminus W) \]

\[ \text{POW}(\cdot; -1) \]

\[ \text{PROD}(\cdot, \cdot) \]

\[ p(V | W) \]

\[ (a) \text{ Pipeline for } \text{COND}(\cdot, W) \]

\[ p(Y | \text{do}(A)) \]

\[ \text{MARG}(\cdot; Z) \]

\[ \text{COND}(\cdot; A \cup Z) \]

\[ MARG(\cdot; X \setminus (A \cup Y \cup Z)) \]

\[ \text{MARG}(\cdot; X \setminus Z) \]

\[ p(X) \]

\[ p(Y) \]

\[ (b) \text{ Pipeline for entire backdoor query} \]

**Problem:** Cannot guarantee that input to POW is deterministic, even if \( p(X) \) is deterministic.
Definition (Marginal Determinism, Choi et al. 2020)

Given a subset of variables \( Q \subseteq X \), a PC is \( Q \)-deterministic if the children of a sum node \( T \) correspond to different values of \( Q \) (for sum nodes with \( sc(T) \cap Q \neq \emptyset \)).
Marginal Determinism

Definition (Marginal Determinism, Choi et al. 2020)

Given a subset of variables $Q \subseteq X$, a PC is $Q$-deterministic if the children of a sum node $T$ correspond to different values of $Q$ (for sum nodes with $\text{sc}(T) \cap Q \neq \emptyset$).

Motivation: If a circuit is marginally deterministic w.r.t $Q$, then we can marginalize out $X \setminus Q$ and obtain a deterministic circuit!

(a) $Q = \{A, Z\}$-deterministic

(b) $Q = \{A, Y\}$-deterministic
Definition (Marginal Determinism, Choi et al. 2020)

Given a subset of variables $Q \subseteq X$, a PC is $Q$-deterministic if the children of a sum node $T$ correspond to different values of $Q$ (for sum nodes with $\text{sc}(T) \cap Q \neq \emptyset$).

Motivation: If a circuit is marginally deterministic w.r.t $Q$, then we can marginalize out $X \setminus Q$ and obtain a deterministic circuit!
If (the circuit encoding) \( p(X) \) is \((A \cup Z)\)-deterministic, then the input to \( \text{POW} \) is guaranteed to be deterministic.
If (the circuit encoding) \( p(X) \) is \((A \cup Z)\)-deterministic, then the input to POW is guaranteed to be deterministic.

(a) Pipeline for \( \text{COND}(\cdot; A \cup Z) \)

(b) Pipeline for entire backdoor query
Tractable Causal Inference

If (the circuit encoding) $p(X)$ is $(A \cup Z)$-deterministic, then the input to $POW$ is guaranteed to be deterministic.

\[ p(Y|A, Z) \]

\[ MARG(\cdot; Y) \]

\[ PROD(\cdot, \cdot) \]

\[ POW(\cdot; -1) \]

\[ p(A, Y, Z) \]

(a) Pipeline for $COND(\cdot, A \cup Z)$

(b) Pipeline for entire backdoor query

$\implies$ all operations are tractable according to Atlas

\[ p(Y|do(A)) \]

\[ MARG(\cdot; Z) \]

\[ PROD(\cdot, \cdot) \]

\[ COND(\cdot; A \cup Z) \]

\[ MARG(\cdot; X \setminus (A \cup Y \cup Z)) \]

\[ p(X) \]
Tractable Causal Inference

If (the circuit encoding) $p(X)$ is $(A \cup Z)$-deterministic, then the input to $POW$ is guaranteed to be deterministic.

\[
p(X) = \begin{cases} \text{MARG}(\cdot; X \setminus (A \cup Y \cup Z)) \\ \text{PROD}(\cdot, \cdot) \\ \text{COND}(\cdot; A \cup Z) \\ p(Y | do(A)) \\ \text{MARG}(\cdot; Z) \\ \text{PROD}(\cdot, \cdot) \\ p(Y | A, Z) \\ \text{MARG}(\cdot; Y) \\ \text{POW}(\cdot; -1) \\ p(A, Y, Z) \end{cases}
\]

(a) Pipeline for $\text{COND}(\cdot, A \cup Z)$

(b) Pipeline for entire backdoor query

\[\Rightarrow\text{ all operations are tractable according to Atlas}\]

\[\Rightarrow\text{ can compute causal effect in } O(|p|^3) \text{ time}\]

(can improve to $O(|p|^2)$)
Open Questions

- Are all causal queries derived by the do-calculus tractable in PTIME (for some non-trivial marginal determinism condition)?
- What is the optimal complexity for these queries?
Questions answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Questions answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
tractability vs expressive efficiency
<table>
<thead>
<tr>
<th>Model</th>
<th>Smooth</th>
<th>Decomposable</th>
<th>Deterministic</th>
<th>Structured Decomposable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum-Product Networks (SPNs)</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Cutset Networks (CNets)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Probabilistic Decision Graphs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>(Affine) ADDs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PSDDs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Low-treewidth PGMs

Tree, polytrees and Thin Junction trees can be turned into
- decomposable
- smooth
- deterministic circuits

Therefore they support tractable
- EVI
- MAR/CON
- MAP
Arithmetic Circuits (ACs)

ACs (Darwiche 2003) are decomposable, smooth, and deterministic. They support tractable EVI, MAR/CON, and MAP.

⇒ parameters are attached to the leaves
⇒ ...but can be moved to the sum node edges (Rooshenas et al. 2014)

Lowd and Rooshenas, “Learning Markov Networks With Arithmetic Circuits”, 2013
Sum-Product Networks (SPNs)

SPNs (Poon et al. 2011) are decomposable, smooth, and deterministic. They support tractable EVI, MAR/CON, and MAP.

deterministic SPNs are also called selective (Peharz et al. 2014)
Cutset Networks (CNets)

CNets (Rahman et al. 2014) are decomposable, smooth, deterministic, and support tractable EVI, MAR/CON, and MAP.

Di Mauro et al., “Learning Accurate Cutset Networks by Exploiting Decomposability”, 2015
Probabilistic Sentential Decision Diagrams

PSDDs \((Kisa \ et \ al. \ 2014)\) are

- structured
- decomposable
- smooth
- deterministic

They support tractable

- EVI
- MAR/CON
- MAP
- Complex queries!

---

Kisa et al., “Probabilistic sentential decision diagrams”, 2014
Shen et al., “Conditional PSDDs: Modeling and learning with modular knowledge”, 2018
Probabilistic Decision Graphs

PDGs \textit{(Jaeger 2004)} are structured, decomposable, smooth, and deterministic. They support tractable EVI, MAR/CON, MAP, and Complex queries!

\textit{Jaeger, “Probabilistic decision graphs—combining verification and AI techniques for probabilistic inference”, 2004}

\textit{Jaeger et al., “Learning probabilistic decision graphs”, 2006}
AndOrGarphs

(Dechter et al. 2007) are
- structured
- decomposable
- smooth
- deterministic

They support tractable
- EVI
- MAR/CON
- MAP
- Complex queries!

Dechter and Mateescu, “AND/OR search spaces for graphical models”, 2007
Marinescu and Dechter, “Best-first AND/OR search for 0/1 integer programming”, 2007
Probabilistic circuits seem awfully general.

Are all tractable probabilistic models probabilistic circuits?
Determinantal Point Processes (DPPs)

DPPs are models where probabilities are specified by (sub)determinants

\[ L = \begin{bmatrix} 1 & 0.9 & 0.8 & 0 \\ 0.9 & 0.97 & 0.96 & 0 \\ 0.8 & 0.96 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]

\[
\Pr_L(X_1 = 1, X_2 = 0, X_3 = 1, X_4 = 0) = \frac{1}{\det(L + I)} \det(L_{\{1,2\}})
\]
Enter: Determinantal Point Processes (DPPs)

DPPs are models where probabilities are specified by (sub)determinants

\[ L = \begin{bmatrix}
1 & 0.9 & 0.8 & 0 \\
0.9 & 0.97 & 0.96 & 0 \\
0.8 & 0.96 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \]

Tractable likelihoods and marginals

Global Negative Dependence

Diversity in recommendation systems

\[ \Pr_L(X_1 = 1, X_2 = 0, X_3 = 1, X_4 = 0) = \frac{1}{\det(L + I)} \det(L_{\{1,2\}}) \]
Are all tractable probabilistic models probabilistic circuits?
Relationship between PCs and DPPs

Probabilistic Circuits

Determinantal Point Processes

Positive Dependence

Fully Factorized

We cannot tractably represent DPPs with subclasses of PCs

We cannot tractably represent DPPs with subclasses of PCs

PSDDs
More Tractable
Fewer Constraints
Deterministic and Decomposable PCs
Deterministic PCs with no negative parameters
Deterministic PCs with negative parameters

We cannot tractably represent DPPs with subclasses of PCs

PSDDs

More Tractable

Deterministic and Decomposable PCs

Deterministic PCs with no negative parameters

Deterministic PCs with negative parameters

Decomposable PCs with no negative parameters (SPNs)

Fewer Constraints

We cannot tractably represent DPPs with subclasses of PCs
Theorem (Martens and Medabalimi, 2014). Let $P_n$ be the uniform distribution over spanning trees on $K_n$. For $n \geq 20$, the size of any smooth and decomposable PC that represents $P_n$ is at least $2^{n/30240}$.

Based on arithmetic circuit lower bounds by Ran Raz and Amir Yehudayoff

Decomposable PCs are **Syntactically Multilinear** Arithmetic Circuits:

Definition 7 (Multilinear Arithmetic Circuit) If every node of an arithmetic circuit $\Phi$ over $y$ computes a multilinear polynomial in $y$, $\Phi$ is said to be a (semantically) multilinear arithmetic circuit. And if for every product node in $\Phi$, the scopes of its child nodes are pair-wise disjoint, $\Phi$ is said to be a syntactically multilinear arithmetic circuit.
**Theorem** (Snell, 1995). The uniform distribution over spanning trees on the complete graph $K_n$ is a DPP over $\binom{n}{2}$ edges.

**Theorem** (Martens and Medabalimi, 2014). Let $P_n$ be the uniform distribution over spanning trees on $K_n$. For $n \geq 20$, the size of any smooth and decomposable PC that represents $P_n$ is at least $2^{n/30240}$. 
Probabilistic Generating Circuits

A Tractable Unifying Framework for PCs and DPPs

Probability Generating Functions

\[
g_\beta = 0.16z_1 z_2 z_3 + 0.04z_1 z_2 + 0.08z_1 z_3 + 0.02z_1
+ 0.48z_2 z_3 + 0.12z_2 + 0.08z_3 + 0.02.
\]

<table>
<thead>
<tr>
<th>(X_1)</th>
<th>(X_2)</th>
<th>(X_3)</th>
<th>(\text{Pr}_{\beta})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.08</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.12</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.48</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.08</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.04</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.16</td>
</tr>
</tbody>
</table>
Probability Generating Functions

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$\Pr_{\beta}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.08</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.12</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.48</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.08</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.04</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.16</td>
</tr>
</tbody>
</table>

$$g_{\beta} = 0.16z_1z_2z_3 + 0.04z_1z_2 + 0.08z_1z_3 + 0.02z_1 + 0.48z_2z_3 + 0.12z_2 + 0.08z_3 + 0.02.$$ 

$$g_{\beta} = (0.1(z_1 + 1)(6z_2 + 1) - 0.4z_1z_2)(0.8z_3 + 0.2).$$
1. Sum nodes with weighted edges to children.
2. Product nodes with unweighted edges to children.
3. Leaf nodes: $z_i$ or constant.

\[ g_\beta = (0.1(z_1 + 1)(6z_2 + 1) - 0.4z_1z_2)(0.8z_3 + 0.2) \]
(Smooth & Decomposable) PCs represents probability mass functions:

\[
m_\beta = 0.16X_1X_2X_3 + 0.04X_1X_2\overline{X}_3 + 0.08X_1\overline{X}_2X_3 + 0.02X_1\overline{X}_2\overline{X}_3 \\
+ 0.48\overline{X}_1X_2X_3 + 0.12\overline{X}_1X_2\overline{X}_3 + 0.08\overline{X}_1\overline{X}_2X_3 + 0.02\overline{X}_1\overline{X}_2\overline{X}_3
\]

PGCs represent probability generating functions:

\[
g_\beta = 0.16z_1z_2z_3 + 0.04z_1z_2 + 0.08z_1z_3 + 0.02z_1 \\
+ 0.48z_2z_3 + 0.12z_2 + 0.08z_1z_3 + 0.02
\]

Given a smooth & decomposable PC, by setting \(\overline{X}_i\) to 1, and \(X_i\) to \(z_i\), we obtain a PGC that represents the PC.
Tractable Likelihood (EVID) or Marginals (MAR)?

Pr(X₁ = 1, X₂ = 0, ... ) =?
PGCs Support Tractable Likelihoods/Marginals

Purely symbolic

$z_i = \begin{cases} t, & X_i = 1 \\ 0, & X_i = 0 \\ 1, & \text{otherwise} \end{cases}$

$\Pr(X_1 = 1, X_2 = 0, \ldots) = ?$

$p(t) = \alpha_k t^k + \cdots + \alpha_1 t$
PGCs Support Tractable Likelihoods/Marginals

Purely symbolic

\[ z_i = \begin{cases} t, & X_i = 1 \\ 0, & X_i = 0 \\ 1, & \text{otherwise} \end{cases} \]

\[ \Pr(X_1 = 1, X_2 = 0, \ldots) = ? \]

\[ p(t) = \alpha_k t^k + \ldots + \alpha_1 t \]

- Monomials setting to true variables that must be false are 0-ed out
- Other monomials contribute to result.
- Only monomials that set all required variables to true have max degree.
- Sum those up

PGCs Support Tractable Likelihoods/Marginals

Purely symbolic

\[ z_i = \begin{cases} 
  t, & X_i = 1 \\
  0, & X_i = 0 \\
  1, & \text{otherwise} 
\end{cases} \]

\[ \Pr(X_1 = 1, X_2 = 0, \ldots) =? \]

\[ p(t) = \alpha_k t^k + \cdots + \alpha_1 t \]

\[ \alpha_k \text{ gives the answer} \]
Example

\[ \Pr(X_2 = 1, X_3 = 0) =? \]

![Diagram with probabilities and nodes](image)
Example

\[ \Pr(X_2 = 1, X_3 = 0) = ? \]
Example

\[ \Pr(X_2 = 1, X_3 = 0) = ? \]

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$\Pr_{\beta}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.08</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.12</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0.48</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.08</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.04</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.16</td>
</tr>
</tbody>
</table>
Inference Time Complexity

Given a PGC of size $m$ (#edges) over $n$ random variables.

Algorithm 1 (Zhang et al., ICML 2021):

Bottom-up pass w/ $z_i = t, 0$ or $1$

Product/sum of degree-$n$ polynomials at each node

$= O(mn^2)$

or $O(mn \log n \log \log n)$
Inference Time Complexity

Given a PGC of size $m$ (#edges) over $n$ random variables.

Algorithm 1 (Zhang et al., ICML 2021):
- Bottom-up pass w/ $z_i = t$, 0 or 1
- Product/sum of degree-$n$ polynomials at each node

$$\implies O(mn^2)$$ or $O(mn \log n \log \log n)$

Algorithm 2 (Harviainen et al., UAI 2023):
- Bottom-up pass w/ $t = 0, 1, \ldots, n$
- Polynomial interpolation at $t = 0, 1, \ldots, n$

$$\implies O(mn)$$
Syntactic vs. Semantic Restrictions

+ PGCs are tractable when semantically multilinear
  + No need for PC decomposability/syntactic multilinearity or other properties...

- Checking Validity of PGCs is Hard

Theorem (Harviainen et al.). It is NP-hard to check if a PGC encodes a valid probability generating polynomial.
DPPs as PGCs

The generating polynomial for a DPP with kernel $L$ is given by:

$$g_L = \frac{1}{\det(L + I)} \det(I + L \text{diag}(z_1, \ldots, z_n)).$$

We need it as a sum of products to obtain a Probabilistic Generating Circuit.
DPPs as PGCs

The generating polynomial for a DPP with kernel $L$ is given by:

$$g_L = \frac{1}{\det(L + I)} \det(I + L \text{diag}(z_1, \ldots, z_n)).$$

We need it as a sum of products to obtain a Probabilistic Generating Circuit.
DPPs as PGCs

The generating polynomial for a DPP with kernel $L$ is given by:

$$g_L = \frac{1}{\det(L + I)} \det(I + L \text{diag}(z_1, \ldots, z_n)).$$

Division-free determinant algorithm
(Samuelson-Berkowitz algorithm)

$g_L$ can be represented as a PGC of size $O(n^4)$. 

## Experiment Results: Amazon Baby Registries

SimplePGC achieves SOTA result on 11/15 datasets


<table>
<thead>
<tr>
<th>Category</th>
<th>DPP</th>
<th>Strudel</th>
<th>EiNet</th>
<th>MT</th>
<th>SimplePGC</th>
</tr>
</thead>
<tbody>
<tr>
<td>bath</td>
<td>-8.55</td>
<td>-8.38</td>
<td>-8.49</td>
<td>-8.53</td>
<td>-8.29*</td>
</tr>
<tr>
<td>bedding</td>
<td>-8.65</td>
<td>-8.50</td>
<td>-8.55</td>
<td>-8.59</td>
<td>-8.41*</td>
</tr>
<tr>
<td>carseats</td>
<td>-4.74</td>
<td>-4.79</td>
<td>-4.72</td>
<td>-4.76</td>
<td>-4.64*</td>
</tr>
<tr>
<td>feeding</td>
<td>-11.86</td>
<td>-11.42</td>
<td>-11.27</td>
<td>-11.30</td>
<td>-11.17*</td>
</tr>
<tr>
<td>furniture</td>
<td>-4.38</td>
<td>-4.39</td>
<td>-4.38</td>
<td>-4.43</td>
<td>-4.34*</td>
</tr>
<tr>
<td>gifts</td>
<td>-3.51</td>
<td><strong>3.39</strong></td>
<td>-3.42</td>
<td>-3.48</td>
<td>-3.47*</td>
</tr>
<tr>
<td>health</td>
<td>-7.40</td>
<td>-7.37</td>
<td>-7.47</td>
<td>-7.49</td>
<td><strong>7.24</strong>*</td>
</tr>
<tr>
<td>media</td>
<td>-8.36</td>
<td><strong>7.62</strong></td>
<td>-7.82</td>
<td>-7.93</td>
<td>-7.69*</td>
</tr>
<tr>
<td>moms</td>
<td>-3.55</td>
<td>-3.52</td>
<td><strong>3.48</strong></td>
<td>-3.54</td>
<td>-3.53*</td>
</tr>
<tr>
<td>safety</td>
<td>-4.28</td>
<td>-4.43</td>
<td>-4.39</td>
<td>-4.36</td>
<td>-4.28***</td>
</tr>
<tr>
<td>strollers</td>
<td>-5.30</td>
<td>-5.07</td>
<td>-5.07</td>
<td>-5.14</td>
<td><strong>5.00</strong>*</td>
</tr>
<tr>
<td>toys</td>
<td>-8.05</td>
<td><strong>7.61</strong></td>
<td>-7.84</td>
<td>-7.88</td>
<td>-7.62*</td>
</tr>
</tbody>
</table>

SimplePGC achieves SOTA result on 11/15 datasets
Beyond DPPs: Strongly Rayleigh Distributions

DPPs are strongly Rayleigh distributions

**Definition.** A probability distribution over binary random variables $X_1, \ldots, X_n$ (or equivalently, subsets of $[n] := \{1, 2, \ldots, n\}$) is strongly Rayleigh if its probability generating polynomial $g$ is real-stable; that is, for $z_i \in \mathbb{C}$, if $\text{Im}(z_i) > 0$ for all $z_i$, then $g(z_1, \ldots, z_n) \neq 0$.

We can efficiently sample from strongly Rayleigh distributions by MCMC (with polynomial bound on mixing time)
Efficient Sampling from SR Distributions

**Theorem** (Li et al., 2016). Let \( \pi \) be a strongly Rayleigh distribution over \([n]\), we can efficiently sample from \( \pi \) by sampling from its symmetric homogenization \( \pi_{sh} \); for \( S \subset [2n] \), define

\[
\pi_{sh}(S) := \begin{cases} 
\pi(S \cap [n]) \binom{n}{|S \cap [n]|}^{-1}, & \text{if } |S| = n \\
0, & \text{otherwise}
\end{cases}
\]

in particular, \( \pi_{sh} \) is also strongly Rayleigh and the mixing time of a Gibbs-exchange sampler with initial set \( S_0 \) is bounded as

\[
\tau(\epsilon) \leq 2n^2 \left( \log \left( \frac{n}{|S_0|} \right) + \log \pi(S_0)^{-1} + \log \epsilon^{-1} \right)
\]
Relationship between PGCs and SR Distributions
Relationship between PGCs and SR Distributions

Compact PGCs

\[0.5 + 0.5 \prod_i \approx_i\]

DPPs

SR Distributions

?
Let $K_{m,n} = (U \cup V, E)$ be a complete bipartite graph, the signed double function generating polynomial is defined as

$$DF_{m,n}(e) = \sum_{F,H} (-1)^{|F|+|H|} \prod_{(i,j) \in F} e_{i,j} \prod_{(i',j') \in H} e_{i',j'}$$

where the sum is taken over all partial functions $U \rightarrow V$ and $V \rightarrow U$, respectively. Each pair of $(F, H)$ is a double function of $K_{m,n}$.

![Figure 4. The thick edges are a matching of size two.](image1)
![Figure 5. The thick edges form a total function $U \rightarrow V$, which is not injective.](image2)
![Figure 6. The thick edges form a partial function from $V$ to $U$.](image3)
![Figure 7. A double function.](image4)
Not All SR Distributions have Compact PGCs (Bläser 2023)

\[ DF_{m,n}(e) = \sum_{F,H} (-1)^{|F|+|H|} \prod_{(i,j) \in F} e_{i,j} \prod_{(i',j') \in H} e_{i',j'} \]

Generalize to bipartite multigraph \( K_{m,n}^{(d)} \)

d: each edge from U to V has d copies

\[ DF_{m,n}(e^{(d)}) = \sum_{F,H} (-1)^{|F|+|H|} \prod_{(i,j) \in F \setminus H} \sum_{\delta=1}^d e_{i,j}^{(\delta)} \prod_{(i',j') \in H \cap F} \sum_{1 \leq \delta' < \gamma \leq d} e_{i',j'}^{(\delta')} e_{i',j'}^{(\gamma)} \prod_{(i'',j'') \in H \setminus F} \sum_{\delta''=1}^d e_{i'',j''}^{(\delta''')} \]

\( DF_{n,n}^{(n+2)} \) is real-stable and its evaluation is \#P-hard.

\( DF_{n,n}^{(n+2)} \) does not define an SR distribution as it has negative coefficients.
Not All SR Distributions have Compact PGCs (Bläser 2023)

**Definition.** For a polynomial $f(z_1, \ldots, z_n)$ with $z_i$ of degree $k_i$, the inversion of $f$ is defined as $\prod_i z_i^{-k_i} f(-1/z_1, \ldots, -1/z_i, \ldots, -1/z_n)$.

The inversion of a real stable polynomial is also real stable.

Let $P_n$ be the inversion of $DF_{n,n}^{(n+2)}$, then $P_n$ is a multilinear and real stable polynomial with all coefficients non-negative.

**Theorem** (Bläser, 2023). Assuming $P^{\#P} \not\subseteq P/Poly$. Let $\hat{P}_n$ be the normalized $P_n$, then $\hat{P}_n$ cannot be represented as polynomial-size PGCs.
Relationship between PGCs and SR Distributions

\[ 0.5 + 0.5 \prod_i z_i \]

\[ \hat{P}_n \]
Probabilistic generating circuits seem awfully general.

Are all tractable probabilistic models probabilistic generating circuits?
Questions answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Questions answered today

1. What are probabilistic queries? Are current models tractable? (Guy)
2. What are probabilistic circuits and why are they tractable? (Guy)
3. What is the connection to logical circuit languages? (YooJung)
4. How do I compile my favorite model into a circuit? (YooJung)
5. How are circuit size and tractability related? (YooJung)
6. What’s the most impressive query we can efficiently compute? (YooJung)
7. Are all tractable distributions probabilistic circuits? (Guy)
8. How to learn probabilistic circuits from data? (Guy)
Building Probabilistic Circuits
Information

Prior Knowledge
- domain assumptions
- constraints
- other models

Data
- experimental data
- samples
- measurements

compilation

learning

Circuits
- decomposability
- smoothness
- determinism
- compatibility

Structure

Parameters
- $\theta$, $w$
- generative
- discriminative
- Bayesian
- credal
Origins: Compilation
Compiling probabilistic graphical models

Arithmetic circuits

(Darwiche 2002, 2003, 2009)

- Compile a given Bayesian network into an arithmetic circuit—a smooth, decomposable and deterministic PCs
- Either via logic encoding of Bayesian network + knowledge compilation
- Or record “execution trace” (sum and product operations) of traditional inference algorithms (junction tree, variable elimination)
Logic circuits, interplay between structural properties and tractable reasoning
(Darwiche et al. 2002a)

Converting probabilistic graphical models via knowledge compilation
(Darwiche 2002)

Logic circuit compilers
(Darwiche 2004; Muise et al. 2012; Bova et al. 2015; Lagniez et al. 2017; Oztok et al. 2018)

Neuro-symbolic models using logic circuits
(Ahmed et al. 2022a,b)
Parameter Learning
Gradient descent (of course)

- PCs are computational graphs
- Hence we can just learn them as any other neural net using SGD
- Use re-parameterization if parameters should satisfy constraints:
  - soft-max for sum-weights (non-negative, sum-to-one)
  - soft-plus for variances
  - low-rank plus diagonal for covariance matrices
- Allows for conditional distributions
Conditional PCs

(Shao et al. 2019)
Maximum likelihood (frequentist)

PCs can be interpreted as *hierarchical latent variable models*, where each sum node corresponds to a discrete latent variable (*Peharz et al. 2016*). This allows to perform classical maximum-likelihood estimation.
Closed-form maximum likelihood

When the circuit is deterministic, there is even an closed-form ML solution, which is incredibly fast:

```julia
julia> using ProbabilisticCircuits;
julia> data, structure = load(...);
julia> num_examples(data)
17412
julia> num_edges(structure)
270448
julia> @btime estimate_parameters(structure, data);
  63.585 ms (1182350 allocations: 65.97 MiB)
```

Custom SIMD and CUDA kernels to parallelize over layers and training examples.

https://github.com/Juice-jl/
When the PC is not deterministic, we can still apply expectation-maximization (Peharz et al. 2016). EM can piggy-back on autodiff:

```
train_x, valid_x, test_x = get_mnist_images([7])

graph = Graph.poon_domingos_structure(shape=(28,28), delta=[7])
args = EinsumNetwork.Args(num_var=train_x.shape[1], num_dims=1,
num_classes=1, num_sums=28,
num_input_distributions=28,
exponential_family=EinsumNetwork.BinomialArray,
exponential_family_args={'N':255},
online_em_frequency=1, online_em_stepsize=0.05)

PC = EinsumNetwork.EinsumNetwork(graph, args)
PC.initialize()
PC.to('cuda')
```

https://github.com/cambridge-mlg/EinsumNetworks
for epoch_count in range(10):
    train_ll, valid_ll, test_ll = compute_loglikelihood()
    start_t = time.time()

    for idx in get_batches(train_x, 100):
        outputs = PC.forward(train_x[idx, :])
        log_likelihood = EinsumNetwork.log_likelihoods(outputs).sum()
        log_likelihood.backward()
        PC.em_process_batch()

    print_performance(epoch_count, train_ll, valid_ll, test_ll, time.time() - start_t)

https://github.com/cambridge-mlg/EinsumNetworks
# train sample: 5175
# parameters: 1573486

<table>
<thead>
<tr>
<th>epoch 0</th>
<th>train LL -140936.80</th>
<th>valid LL -140955.72</th>
<th>test LL -141033.80</th>
<th>... elapsed time 3.621 sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>epoch 1</td>
<td>train LL -15916.14</td>
<td>valid LL -15693.25</td>
<td>test LL -15976.43</td>
<td>... elapsed time 3.438 sec</td>
</tr>
<tr>
<td>epoch 2</td>
<td>train LL -10865.67</td>
<td>valid LL -10616.72</td>
<td>test LL -10943.56</td>
<td>... elapsed time 3.436 sec</td>
</tr>
<tr>
<td>epoch 3</td>
<td>train LL -10388.53</td>
<td>valid LL -10158.84</td>
<td>test LL -10475.49</td>
<td>... elapsed time 3.473 sec</td>
</tr>
<tr>
<td>epoch 4</td>
<td>train LL -10264.11</td>
<td>valid LL -10041.66</td>
<td>test LL -10352.59</td>
<td>... elapsed time 3.497 sec</td>
</tr>
<tr>
<td>epoch 5</td>
<td>train LL -10212.66</td>
<td>valid LL -10001.09</td>
<td>test LL -10319.35</td>
<td>... elapsed time 3.584 sec</td>
</tr>
<tr>
<td>epoch 6</td>
<td>train LL -10192.21</td>
<td>valid LL -9965.98</td>
<td>test LL -10314.84</td>
<td>... elapsed time 3.508 sec</td>
</tr>
<tr>
<td>epoch 7</td>
<td>train LL -10153.97</td>
<td>valid LL -9920.09</td>
<td>test LL -10261.41</td>
<td>... elapsed time 3.446 sec</td>
</tr>
<tr>
<td>epoch 8</td>
<td>train LL -10112.95</td>
<td>valid LL -9882.48</td>
<td>test LL -10236.34</td>
<td>... elapsed time 3.579 sec</td>
</tr>
<tr>
<td>epoch 9</td>
<td>train LL -10093.31</td>
<td>valid LL -9862.15</td>
<td>test LL -10200.94</td>
<td>... elapsed time 3.483 sec</td>
</tr>
</tbody>
</table>

---

**Peharz et al., “Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits”, 2020**
Structure Learning
Region graphs

Laying out the PC structure on a high level

- Region graphs (RGs) describe decompositional structure
- RGs are bipartite, directed graphs containing regions ($\mathcal{R}$) and partitions ($\mathcal{P}$)
- Input and output nodes of the RG are regions
- Regions have a scope (receptive field), denoted as $sc(\mathcal{R}) \subseteq X$
- For every partition $\mathcal{P}$ it holds that

$$sc(\mathcal{R}_{out}) = \bigcup_{\mathcal{R}_{in} \in inputs(\mathcal{P})} sc(\mathcal{R}_{in})$$

$$sc(\mathcal{R'}) \cap sc(\mathcal{R''}) = \emptyset, \quad \mathcal{R'} \neq \mathcal{R''} \in inputs(\mathcal{P})$$
Example region graph

(Here, every partition has 2 input regions. This is often assumed, but not necessary.)
From region graphs to PCs
From region graphs to PCs

Equip each input region with non-linear units
\[ \phi_1, \ldots, \phi_K \]
From region graphs to PCs

Equip each internal region with sum nodes
From region graphs to PCs

Often, output region has only a single sum.
From region graphs to PCs

Equip partitions with products, combining units in input regions in all possible ways
Equip partitions with products, combining units in input regions in all possible ways.
From region graphs to PCs

Connect products to sum units above
From region graphs to PCs

- Equip each input region (leaf) $\mathcal{R}$ with $K$ units $\phi_1, \ldots, \phi_K$, which are non-linear functions over $sc(\mathcal{R})$. Usually, $\phi_1, \ldots, \phi_K$ are probability densities. $K$ can be different for each input region.

- Equip each other region with $K$ sum units. $K$ can be different for each internal region. Often, for the root region $K = 1$, when PC is used as density estimator.

- Equip each partition $\mathcal{P}$ with as many products as there are combinations of units in the input regions to $\mathcal{P}$, selecting one unit from each region. Formally, if $\mathcal{P}$ has input regions $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_I$, insert one product $\prod_{i=1}^I u_i$ for each $(u_1, u_2, \ldots, u_I) \in \mathcal{R}_1 \times \mathcal{R}_2 \times \cdots \times \mathcal{R}_I$.

- Connect each $\prod_{i=1}^I u_i$ in $\mathcal{P}$ to all sum units in the output regions of $\mathcal{P}$. 
From region graphs to PCs

- Resulting PC has alternating sum and product units (not a strong constraint)
- We can easily scale the PC (overparameterize, increase expressivity) by equipping regions with more units
- RGs can be seen as a vectorized version of PCs – each region and partition can be seen as a module
- Resulting PC will be smooth and decomposable, i.e., we can integrate, marginalize, and take conditionals
- After the PC has been constructed, we might discard the RG
Scaling up image models

Latent Variable Distillation

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LVD (ours)</th>
<th>HCLT</th>
<th>EiNet</th>
<th>RAT-SPN</th>
<th>Glow</th>
<th>RealNVP</th>
<th>BIVA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageNet32</td>
<td>4.38</td>
<td>4.82</td>
<td>5.63</td>
<td>6.90</td>
<td>4.09</td>
<td>4.28</td>
<td>3.96</td>
</tr>
<tr>
<td>ImageNet64</td>
<td>4.12</td>
<td>4.67</td>
<td>5.69</td>
<td>6.82</td>
<td>3.81</td>
<td>3.98</td>
<td>-</td>
</tr>
<tr>
<td>CIFAR</td>
<td>4.37</td>
<td>4.61</td>
<td>5.81</td>
<td>6.95</td>
<td>3.35</td>
<td>3.49</td>
<td>3.08</td>
</tr>
</tbody>
</table>

How to construct and learn RGs?
Generating a random region graph, by recursively splitting $X$ into two random parts:
Image-tailored circuit structure

“Recursive image slicing” (Poon et al. 2011)

Images yield a natural region graph by using axis-aligned splits:

- Start with the full image (=output region)
- Define partitions by applying horizontal and vertical splits
- Recurse on the newly generated sub-images (internal regions)
- Structure somewhat reminiscent to convolutions
- Generates RGs which are “true DAGs,” i.e. regions get re-used
Data-driven structure learning

“Recursive data slicing”

(Gens et al. 2013)

Expand regions with clustering
Data-driven structure learning

“Recursive data slicing” (Gens et al. 2013)

Number of clusters = number of partitions
Try to find independent groups of variables (e.g. independence tests)
Data-driven structure learning

“Recursive data slicing” (Gens et al. 2013)

Success → partition into new regions
Data-driven structure learning

“Recursive data slicing” (Gens et al. 2013)

Try to find independent groups of variables (e.g. independence tests)
Data-driven structure learning

"Recursive data slicing"

(Gens et al. 2013)

Success → partition into new regions

$X_1 \ X_2 \ X_3 \ X_4 \ X_5$
Data-driven structure learning

“Recursive data slicing” (Gens et al. 2013)

Single variable
Data-driven structure learning

“Recursive data slicing” (Gens et al. 2013)

Single variable → input region
Data-driven structure learning

"Recursive data slicing"

(Gens et al. 2013)

Expand regions with clustering
Data-driven structure learning

“Recursive data slicing” (Gens et al. 2013)

Number of clusters = number of partitions

And so on...
Data-driven structure learning

"Recursive data slicing"

- Stopping conditions: minimal number of features, samples, depth, ...
- Clustering ratios also deliver (initial) parameters
- Smooth & Decomposable Circuits
- Tractable integration

(Gens et al. 2013)
LearnSPN

Selected references

- **ID-SPN** *(Rooshenas et al. 2014)*
- **LearnSPN-b/T/B** *(Vergari et al. 2015)*
- For **heterogeneous data** *(Molina et al. 2018)*
- Using **k-means** *(Butz et al. 2018)* or **SVD** splits *(Adel et al. 2015)*
- Learning **DAGs** *(Dennis et al. 2015; Jaini et al. 2018)*
- Approximating independence tests *(Di Mauro et al. 2018)*
Besides clustering, **decision tree learning** can be used as PC learner. **Cutset networks**, decision trees over simple probabilistic models (Chow-Liu trees) *(Rahman et al. 2014)*:

Cutset networks can easily be converted into **smooth, decomposable and deterministic PCs**.
Decision trees as PCs

Also vanilla decision tree learners can be used to learn PCs, by augmenting the leaves with distributions over inputs (Correia et al. 2020). Allows to treat missing features and outlier detection.
Information  Prior Knowledge  Data
domain assumptions  constraints  experimental data
other models

compilation  learning

tables

Circuits  Structure  Parameters
decomposability  \( \theta, w \)  generative
domain smoothness  discriminative
determinism  Bayesian
compatibility  credal


References II


- Darwiche, Adnan (2003). “A Differential Approach to Inference in Bayesian Networks”. In: J.ACM.


References III


References V


References VII


References VIII


**References XI**


References XII

- Vlastelica, Marin, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolínek (2020). “Differentiation of blackbox combinatorial solvers”. In: ICLR.


