Computational Probabilistic Models

Guy Van den Broeck

Perspectives in AI - Mar 31, 2022
What is the right abstraction for distributions?

Probabilistic graphical models is how we do probabilistic AI!

Graphical models of variable-level (in)dependence are a broken abstraction.
What is the right abstraction for distributions?

Probabilistic graphical models is how we do probabilistic AI!

Graphical models of variable-level (in)dependence are a broken abstraction.

3.14 \( \text{Smokes}(x) \land \text{Friends}(x,y) \Rightarrow \text{Smokes}(y) \)
What is the right abstraction for distributions?

Probabilistic graphical models is how we do probabilistic AI!

Graphical models of variable-level (in)dependence are a broken abstraction.

Bean Machine

\[ \mu_k \sim \text{Normal}(\alpha, \beta) \]
\[ \sigma_k \sim \text{Gamma}(\nu, \rho) \]
\[ \theta_k \sim \text{Dirichlet}(\kappa) \]
\[ x_i \sim \begin{cases} 
\text{Categorical}(\text{init}) & \text{if } i = 0 \\
\text{Categorical}(\theta_{x_{i-1}}) & \text{if } i > 0 
\end{cases} \]
\[ y_i \sim \text{Normal}(\mu_{x_i}, \sigma_{x_i}) \]

[Tehrani et al. PGM20]
Computational Abstractions

Let us think of probability as something that is computed.

Abstraction = Structure of Computation

Two levels of abstraction:

- Probabilistic Programs — “High-level code”
- Probabilistic Circuits — “Machine code”
Probabilistic Circuits
Intractable and tractable models
a **unifying framework** for tractable models
"Every talk needs a joke and a literature overview slide, not necessarily distinct"
- after Ron Graham
Probabilistic circuits

*computational graphs* that recursively define distributions

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

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

⇒ *mixtures*

⇒ *factorizations*
Likelihood \[ p(X_1 = -1.85, X_2 = 0.5, X_3 = -1.3, X_4 = 0.2) \]
Likelihood

\[ p(X_1 = -1.85, X_2 = 0.5, X_3 = -1.3, X_4 = 0.2) \]
Likelihood

\( p(X_1 = -1.85, X_2 = 0.5, X_3 = -1.3, X_4 = 0.2) \)
A sum node is *smooth* if its children depend on the same set of variables.

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

**Tractable marginals**

Darwiche and Marquis, “A Knowledge Compilation Map”, 2002
If $p(x) = \sum_i w_i p_i(x)$, (smoothness):

$$\int p(x) dx = \int \sum_i w_i p_i(x) dx =$$

$$= \sum_i w_i \int p_i(x) dx$$

$\Rightarrow$ integrals are “pushed down” to children

[Darwiche & Marquis JAIR 2001, Poon & Domingos UAI11]
If $p(x, y, z) = p(x)p(y)p(z)$, (decomposability):

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

$\Rightarrow$ integrals decompose into easier ones
**Smoothness** + **decomposability** = **tractable MAR**

Forward pass evaluation for MAR

⇒ *linear in circuit size!*

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

- 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)
Learning Expressive Probabilistic Circuits

Hidden Chow-Liu Trees

Learned CLT structure captures strong pairwise dependencies

6 variables
Learning Expressive Probabilistic Circuits

Hidden Chow-Liu Trees

Learned CLT structure captures strong pairwise dependencies

Learned HCLT structure

Compile into an equivalent PC

Mini-batch Stochastic Expectation Maximization

Lossless Data Compression

Data $\rightarrow$ Encode $\rightarrow$ Bitstream $\rightarrow$ Decode $\rightarrow$ Reconstructed data

Expressive probabilistic model $p(x)$ + Efficient coding algorithm

Determines the theoretical limit of compression rate

How close we can approach the theoretical limit

A Typical Streaming Code – Arithmetic Coding

We want to compress a set of variables (e.g., pixels, letters) \( \{x_1, x_2, \ldots, x_k\} \)

Compress \( x_1 \) with \( - \log p(x_1) \) bits
Compress \( x_2 \) with \( - \log p(x_2|x_1) \) bits
Compress \( x_3 \) with \( - \log p(x_3|x_1, x_2) \) bits

Need to compute

\[
\begin{align*}
p(X_1 < x_1) \\
p(X_1 \leq x_1) \\
p(X_2 < x_2|x_1) \\
p(X_2 \leq x_2|x_1) \\
p(X_3 < x_3|x_1, x_2) \\
p(X_3 \leq x_3|x_1, x_2) \\
\vdots
\end{align*}
\]
Probabilistic Circuits
- Expressive → SoTA likelihood on MNIST.
- Fast → Time complexity of en/decoding is $O(|p| \log(D))$, where $D$ is the # variables and $|p|$ is the size of the PC.
## Lossless Neural Compression with Probabilistic Circuits

### SoTA compression rates

<table>
<thead>
<tr>
<th>Dataset</th>
<th>HCLT (ours)</th>
<th>IDF</th>
<th>BitSwap</th>
<th>BB-ANS</th>
<th>JPEG2000</th>
<th>WebP</th>
<th>McBits</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST</td>
<td>1.24 (1.20)</td>
<td>1.96 (1.90)</td>
<td>1.31 (1.27)</td>
<td>1.42 (1.39)</td>
<td>3.37</td>
<td>2.09 (1.98)</td>
<td></td>
</tr>
<tr>
<td>FashionMNIST</td>
<td>3.37 (3.34)</td>
<td>3.50 (3.47)</td>
<td><strong>3.35</strong> (3.28)</td>
<td>3.69 (3.66)</td>
<td>3.93</td>
<td>4.62 (3.72)</td>
<td></td>
</tr>
<tr>
<td>EMNIST (Letter)</td>
<td><strong>1.84</strong> (1.80)</td>
<td>2.02 (1.95)</td>
<td>1.90 (1.84)</td>
<td>2.29 (2.26)</td>
<td>3.62</td>
<td>3.31 (3.12)</td>
<td></td>
</tr>
<tr>
<td>EMNIST (ByClass)</td>
<td><strong>1.89</strong> (1.85)</td>
<td>2.04 (1.98)</td>
<td>1.91 (1.87)</td>
<td>2.24 (2.23)</td>
<td>3.61</td>
<td>3.34 (3.14)</td>
<td></td>
</tr>
</tbody>
</table>

### Compress and decompress 5-40x faster than NN methods with similar bitrates

<table>
<thead>
<tr>
<th>Method</th>
<th># parameters</th>
<th>Theoretical bpd</th>
<th>Codeword bpd</th>
<th>Comp. time (s)</th>
<th>Decomp. time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC (HCLT, $M = 16$)</td>
<td>3.3M</td>
<td>1.26</td>
<td>1.30</td>
<td>9</td>
<td>44</td>
</tr>
<tr>
<td>PC (HCLT, $M = 24$)</td>
<td>5.1M</td>
<td>1.22</td>
<td>1.26</td>
<td>15</td>
<td>86</td>
</tr>
<tr>
<td>PC (HCLT, $M = 32$)</td>
<td>7.0M</td>
<td>1.20</td>
<td>1.24</td>
<td>26</td>
<td>142</td>
</tr>
<tr>
<td>IDF</td>
<td>24.1M</td>
<td>1.90</td>
<td>1.96</td>
<td>288</td>
<td>592</td>
</tr>
<tr>
<td>BitSwap</td>
<td>2.8M</td>
<td>1.27</td>
<td>1.31</td>
<td>578</td>
<td>326</td>
</tr>
</tbody>
</table>
Lossless Neural Compression with Probabilistic Circuits

Can be effectively combined with Flow models to achieve better generative performance

<table>
<thead>
<tr>
<th>Model</th>
<th>CIFAR10</th>
<th>ImageNet32</th>
<th>ImageNet64</th>
</tr>
</thead>
<tbody>
<tr>
<td>RealNVP</td>
<td>3.49</td>
<td>4.28</td>
<td>3.98</td>
</tr>
<tr>
<td>Glow</td>
<td>3.35</td>
<td>4.09</td>
<td>3.81</td>
</tr>
<tr>
<td>IDF</td>
<td>3.32</td>
<td>4.15</td>
<td>3.90</td>
</tr>
<tr>
<td>IDF++</td>
<td>3.24</td>
<td>4.10</td>
<td>3.81</td>
</tr>
<tr>
<td>PC+IDF</td>
<td>3.28</td>
<td><strong>3.99</strong></td>
<td><strong>3.71</strong></td>
</tr>
</tbody>
</table>
Expressive models without compromises
Prediction with Missing Features

Train Classifier

Test with missing features
Expected Predictions

Consider **all possible complete inputs** and **reason** about the expected behavior of the classifier

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

\(x^o = \text{observed features}\)
\(x^m = \text{missing features}\)

**Experiment:**
- \(f(x) = \) logistic regres.
- \(p(x) = \) naive Bayes

[Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]
Probabilistic Circuits for Missing Data

[Graphs showing accuracy and RMSE for different datasets and methods, including MNIST, FMNIST, Abalone, Delta, and Insurance]

[Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]
Tractable Computation of Expected Kernels

- How to compute the expected kernel given two distributions $p$, $q$?

$$\mathbb{E}_{x \sim p, x' \sim q}[k(x, x')]$$

- Circuit representation for kernel functions, e.g., $k(x, x') = \exp\left(-\sum_{i=1}^{d} |X_i - X_i'|^2\right)$
Tractable Computation of Expected Kernels: Applications

- Reasoning about support vector regression (SVR) with missing features

\[ \mathbb{E}_{x_m \sim p(X_m | x_o)} \left[ \sum_{i=1}^{m} w_i k(x_i, x) + b \right] \]

missing features \hspace{1cm} SVR model

- Collapsed Black-box Importance Sampling: minimize kernelized Stein discrepancy

importance weights

\[ w^* = \arg\min_w \left\{ w^\top K_{p,s} w \left| \sum_{i=1}^{n} w_i = 1, w_i \geq 0 \right. \right\} \]

expected kernel matrix

Model-Based Algorithmic Fairness: FairPC

Learn classifier given
- features \( S \) and \( X \)
- training labels/decisions \( D \)

Group fairness by demographic parity:

*Fair decision* \( D_f\) *should be independent of the sensitive attribute* \( S\)

Discover the **latent fair decision** \( D_f\) by learning a PC.

[Choi et al. AAAI21]
Probabilistic Sufficient Explanations

**Goal**: explain an instance of classification (a specific prediction)

Explanation is a subset of features, s.t.

1. The explanation is “probabilistically sufficient”
   
   Under the feature distribution, given the explanation, the classifier is likely to make the observed prediction.

2. It is minimal and “simple”

[Khosravi et al. IJCAI19, Wang et al. XXAI20]
Queries as pipelines: KLD

$$\text{KLD}(p || q) = \int p(x) \times \log((p(x)/q(x))) \, dx$$
Queries as pipelines: Cross Entropy

$$H(p, q) = \int p(x) \times \log(q(x))dX$$

$$⇒ \text{we can reuse the operations!}$$
Determinism

A sum node is *deterministic* if only one of its children outputs non-zero for any input

$\Rightarrow$ allows *tractable* MAP inference

$\arg\max_x p(x)$

*Deterministic circuit*
<table>
<thead>
<tr>
<th>Operation</th>
<th>( \log(p) )</th>
<th>Input conditions</th>
<th>Output conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOG</td>
<td>Sm, Dec, Det</td>
<td>Sm, Dec</td>
<td>smooth, decomposable, deterministic</td>
</tr>
</tbody>
</table>
# Tractable circuit operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input properties</th>
<th>Tractability</th>
<th>Hardness</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM</td>
<td>$\theta_1 p + \theta_2 q$</td>
<td>(+Cmp)</td>
<td>(+SD)</td>
</tr>
<tr>
<td>PRODUCT</td>
<td>$p \cdot q$</td>
<td>Cmp (+Det, +SD)</td>
<td>Dec (+Det, +SD)</td>
</tr>
<tr>
<td>POWER</td>
<td>$p^n$, $n \in \mathbb{N}$</td>
<td>SD (+Det)</td>
<td>SD (+Det)</td>
</tr>
<tr>
<td></td>
<td>$p^\alpha$, $\alpha \in \mathbb{R}$</td>
<td>Sm, Dec, Det (+SD)</td>
<td>Sm, Dec, Det (+SD)</td>
</tr>
<tr>
<td>QUOTIENT</td>
<td>$p/q$</td>
<td>Cmp; $q$ Det (+Det, +SD)</td>
<td>Dec (+Det, +SD)</td>
</tr>
<tr>
<td>LOG</td>
<td>$\log(p)$</td>
<td>Sm, Dec, Det</td>
<td>Sm, Dec</td>
</tr>
<tr>
<td>EXP</td>
<td>$\exp(p)$</td>
<td>linear</td>
<td>SD</td>
</tr>
</tbody>
</table>

Inference by tractable operations

systematically derive tractable inference algorithm of complex queries

<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, Dec, Det*</td>
<td>coNP-hard w/o SD</td>
</tr>
<tr>
<td><strong>Kullback-Leibler Div.</strong></td>
<td>Sm, SD, Det*</td>
<td>coNP-hard w/o SD</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, Det</td>
<td>#P-hard w/o Det</td>
</tr>
<tr>
<td><strong>Squared Loss</strong></td>
<td>Cmp</td>
<td>#P-hard w/o Cmp</td>
</tr>
</tbody>
</table>
Even harder queries

**Marginal MAP**

Given a set of query variables $Q \subseteq X$ and evidence $e$, find: $\text{argmax}_q p(q|e)$

$\Rightarrow$ *i.e. MAP of a marginal distribution on* $Q$

| NP<sup>pp</sup>-complete for PGMs |
| NP-hard even for PCs tractable for marginals, MAP & entropy |
Pruning circuits

Any parts of circuit not relevant for MMAP state can be pruned away

e.g. $p(X_1 = 1, X_2 = 0)$

We can find such edges in linear time
Iterative MMAP solver

<table>
<thead>
<tr>
<th>Dataset</th>
<th>runtime (# solved)</th>
<th>pruning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLTCS</td>
<td>0.01 (10)</td>
<td>0.63 (10)</td>
</tr>
<tr>
<td>MSNBC</td>
<td>0.03 (10)</td>
<td>0.73 (10)</td>
</tr>
<tr>
<td>KDD</td>
<td>0.04 (10)</td>
<td>0.68 (10)</td>
</tr>
<tr>
<td>Plants</td>
<td>2.95 (10)</td>
<td>2.72 (10)</td>
</tr>
<tr>
<td>Audio</td>
<td>2041.33 (6)</td>
<td>13.70 (10)</td>
</tr>
<tr>
<td>Jester</td>
<td>2913.04 (2)</td>
<td>14.74 (10)</td>
</tr>
<tr>
<td>Netflix</td>
<td>– (0)</td>
<td>47.18 (10)</td>
</tr>
<tr>
<td>Accidents</td>
<td>109.56 (10)</td>
<td>15.86 (10)</td>
</tr>
<tr>
<td>Retail</td>
<td>0.06 (10)</td>
<td>0.81 (10)</td>
</tr>
<tr>
<td>Pumsb-star</td>
<td>2208.27 (7)</td>
<td>20.88 (10)</td>
</tr>
<tr>
<td>DNA</td>
<td>– (0)</td>
<td>505.75 (9)</td>
</tr>
<tr>
<td>Kosarek</td>
<td>48.74 (10)</td>
<td>3.41 (10)</td>
</tr>
<tr>
<td>MSWeb</td>
<td>1543.49 (10)</td>
<td>1.28 (10)</td>
</tr>
<tr>
<td>Book</td>
<td>– (0)</td>
<td>46.50 (10)</td>
</tr>
<tr>
<td>EachMovie</td>
<td>– (0)</td>
<td>1216.89 (8)</td>
</tr>
<tr>
<td>WebKB</td>
<td>– (0)</td>
<td>575.68 (10)</td>
</tr>
<tr>
<td>Reuters-52</td>
<td>– (0)</td>
<td>120.58 (10)</td>
</tr>
<tr>
<td>20 NewsGrp.</td>
<td>– (0)</td>
<td>504.52 (9)</td>
</tr>
<tr>
<td>BBC</td>
<td>– (0)</td>
<td>2757.18 (3)</td>
</tr>
<tr>
<td>Ad</td>
<td>– (0)</td>
<td>1254.37 (8)</td>
</tr>
</tbody>
</table>
tractability is a spectrum
Learn more about probabilistic circuits?

Tutorial (3h)

Probabilistic Circuits

Inference Representations Learning Theory

Antonio Vergari
University of California, Los Angeles
Robert Peharz
TU Eindhoven
YooJung Choi
University of California, Los Angeles
Guy Van den Broeck
University of California, Los Angeles

September 14th, 2020 - Ghent, Belgium - ECML-PKDD 2020

https://youtu.be/2RAG5-L9R70

Overview Paper (80p)

Probabilistic Circuits:
A Unifying Framework for Tractable Probabilistic Models*

YooJung Choi
Antonio Vergari
Guy Van den Broeck
Computer Science Department
University of California
Los Angeles, CA, USA

Contents

1 Introduction 3
2 Probabilistic Inference: Models, Queries, and Tractability 4
   2.1 Probabilistic Models ........................................ 5
   2.2 Probabilistic Queries ....................................... 6
   2.3 Tractable Probabilistic Inference ....................... 8
   2.4 Properties of Tractable Probabilistic Models ....... 9

Probabilistic Programs
Motivation from the AI side:
Making modern AI systems is too hard

System Builders

Model Builders
AI System Builder

Need to integrate uncertainty over the whole system

20% chance of obstacle!

94% chance of obstacle!

99% certain about current location

Inside the Self-Driving Tesla Fatal Accident

The accident may have happened in part because the crash-avoidance system is designed to engage only when radar and computer vision systems agree that there is an obstacle, according to an industry executive with direct...
“When you have the flu you have a cough 70% of the time”

“What is the probability that a patient with a fever has the flu?”

“Routers fail on average every 5 years”

“What is the probability that my packet will reach the target server?”
[SGTVV SIGCOMM’20]
Probabilistic Programs

```plaintext
let x = flip 0.5 in
let y = flip 0.7 in
let z = x || y in
let w = if z then
    my_func(x, y)
else
    ...
in
observe(z)
```

- `let x = flip 0.5 in` means “flip a coin, and output true with probability ½”
- `let y = flip 0.7 in` means “flip a coin, and output true with probability 0.7”
- `let z = x || y in` means “z is true if x or y is true”
- `let w = if z then
    my_func(x, y)
else
    ...` means “if z is true, execute my_func(x, y) otherwise execute ...”
- `in observe(z)` means “reject this execution if z is not true”

Standard (functional) programming constructs: let, if, ...
Why Probabilistic Programming?

- PPLs are proliferating
  
  Pyro  
  Edward  
  HackPPL  
  Stan  
  Figaro

  Venture, Church, IBAL, WebPPL, Infer.NET, Tensorflow Probability, ProbLog, PRISM, LPADs, CPLS, CLP(BN), ICL, PHA, Primula, Storm, Gen, PRISM, PSI, Bean Machine, etc.  
  ... and many many more

- Programming languages are humanity’s biggest knowledge representation achievement!
- Programs should be AI models
Focus on Discrete Models

1. Real programs have inherently discrete structure (e.g. if-statements)
2. Discrete structure is inherent in many domains (graphs, text, ranking, etc.)
3. Many existing PPLs assume smooth and differentiable densities and do not handle discreteness well.

Discrete probabilistic programming is the important unsolved open problem!
Dice language for discrete probabilistic programs

http://dicelang.cs.ucla.edu/ [Holtzen et al. OOPSLA20]

dice is a probabilistic programming language focused on fast exact inference for discrete probabilistic programs. For more information on dice, see the about page.

Below is an online dice code demo. To run the example code, press the "Run" button.

```
fun sendChar(key: int(2), observation: int(2)) {
  let gen = discrete(0.1, 0.25, 0.125, 0.125) in // sample a Foolang character
  let enc = key | gen in
  observe observation == enc
}

// sample a uniform random key: A-9, B-1, C-2, D-3
let key = discrete(0.25, 0.25, 0.25, 0.25) in
// observe the ciphertext CCCC
let tmp = sendChar(key, int(2)) in
let tmp = sendChar(key, int(2)) in
let tmp = sendChar(key, int(2)) in
key
```
Network Verification in Dice

fun n1(init: bool) {
    let l1succeed = flip 0.99 in
    let l2succeed = flip 0.91 in
    init && l1succeed && l2succeed
}

fun n2(init: bool) {
    let routeChoice = flip 0.5 in
    if routeChoice then
        init && flip 0.88 && flip 0.93
    else
        init && flip 0.19 && flip 0.33
}

fun n2(n2(n2(n1(true))))

ECMP equal-cost path protocol: choose randomly which router to forward to

Main routine, combines the networks
Network Verification in Dice

fun n1(init: bool) {
    let l1succeed = flip 0.99 in
    let l2succeed = flip 0.91 in
    init && l1succeed && l2succeed
}

fun n2(init: bool) {
    let routeChoice = flip 0.5 in
    if routeChoice then
        init && flip 0.88 && flip 0.93
    else
        init && flip 0.19 && flip 0.33
}

n2(n2(n1(true)))

ECMP equal-cost path protocol: choose randomly which router to forward to

Main routine, combines the networks

This doesn’t show all the language features of dice:
- Integers
- Tuples
- Bounded recursion
- Bayesian conditioning
- …
Probabilistic Program Inference

\[0.99 \times 0.91 \times 0.5 \times 0.88 \times 0.93 \times 0.5 \times 0.88 \times 0.93\]

+ \[0.99 \times 0.91 \times 0.5 \times 0.19 \times 0.33 \times 0.5 \times 0.88 \times 0.93\]

+ \[\ldots\]
Probabilistic Program Inference

Path enumeration: find all of them!

- Dice
- Dice (Inline)
- Psi
- Psi DP
- WebPPL Exact
- Rejection

Graphs showing time (ms) vs.
- # Characters
- Length
- Length
- Length
Key to Fast Inference: **Factorization** (product nodes)

Easy to see on the graph structure ... how about on the program?
Symbolic Compilation in Dice

• Construct Boolean formula
• Satisfying assignments ≈ paths
• Variables are flips
• Associate weights with flips
• Compile factorized circuit

\[ f_1 f_2 f_4 \lor f_1 \bar{f}_2 f_5 \lor \bar{f}_1 f_3 f_4 \lor \bar{f}_1 \bar{f}_3 f_5 \]

1. let \( x = \text{flip}_1 \ 0.1 \ \text{in} \)
2. let \( y = \text{if} \ x \ \text{then} \ \text{flip}_2 \ 0.2 \ \text{else} \)
   \( \text{flip}_3 \ 0.3 \ \text{in} \)
3. let \( z = \text{if} \ y \ \text{then} \ \text{flip}_4 \ 0.4 \ \text{else} \)
   \( \text{flip}_5 \ 0.5 \ \text{in} \ z \)
Symbolic Compilation in Dice to Probabilistic Circuits

- Probabilistic Program
- Symbolic Compilation
- Weighted Boolean Formula
- Weighted Model Count
- Probabilistic Circuit

Circuit compilation

Logic Circuit (BDD)
Experimental Evaluation

- Example from text analysis: breaking a Caesar cipher

- Competitive with specialized Bayesian network solvers

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Psi (ms)</th>
<th>DP (ms)</th>
<th>Dice (ms)</th>
<th># Parameters</th>
<th># Paths</th>
<th>BDD Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancer</td>
<td>772</td>
<td>46</td>
<td>13</td>
<td>10</td>
<td>$1.1 \times 10^3$</td>
<td>28</td>
</tr>
<tr>
<td>Survey</td>
<td>2477</td>
<td>152</td>
<td>13</td>
<td>21</td>
<td>$1.3 \times 10^4$</td>
<td>73</td>
</tr>
<tr>
<td>Alarm</td>
<td>x</td>
<td>x</td>
<td>25</td>
<td>509</td>
<td>$1.0 \times 10^3$</td>
<td>$1.3 \times 10^3$</td>
</tr>
<tr>
<td>Insurance</td>
<td>x</td>
<td>x</td>
<td>212</td>
<td>984</td>
<td>$1.2 \times 10^4$</td>
<td>$1.0 \times 10^5$</td>
</tr>
<tr>
<td>Hepar2</td>
<td>x</td>
<td>x</td>
<td>54</td>
<td>48</td>
<td>$2.9 \times 10^6$</td>
<td>$1.3 \times 10^3$</td>
</tr>
<tr>
<td>Hailfinder</td>
<td>x</td>
<td>x</td>
<td>618</td>
<td>2656</td>
<td>$2.0 \times 10^7$</td>
<td>$6.5 \times 10^4$</td>
</tr>
<tr>
<td>Pigs</td>
<td>x</td>
<td>x</td>
<td>72</td>
<td>5618</td>
<td>$7.3 \times 10^{9}$</td>
<td>35</td>
</tr>
<tr>
<td>Water</td>
<td>x</td>
<td>x</td>
<td>2590</td>
<td>$1.0 \times 10^4$</td>
<td>$3.2 \times 10^5$</td>
<td>$5.1 \times 10^4$</td>
</tr>
<tr>
<td>Munin</td>
<td>x</td>
<td>x</td>
<td>1866</td>
<td>$8.1 \times 10^5$</td>
<td>$2.1 \times 10^{1622}$</td>
<td>$1.1 \times 10^4$</td>
</tr>
</tbody>
</table>
Better Inference. How?

Exploit modularity - program structure

1. **AI modularity:**
   Discover contextual independencies and **factorize**

2. **PL modularity:**
   Compile procedure summaries and reuse at each call site

Reason about programs!  Compiler optimizations:

3. **Flip hoisting optimization**

4. **Determinism, optimize integer representation, etc.**
Flip Hoisting

- Fewer flips = smaller compiled circuits = faster
- But, be careful with soundness:

```
let x = flip 0.1 in let z = flip 0.2 in
let y = if x && z then flip 0.3
    else if x && !z then flip 0.4
    else flip 0.3
in y
```

≡

```
let x = flip 0.1 in let z = flip 0.2 in
let tmp = flip 0.3 in
let y = if x && z then tmp
    else if x && !z then flip 0.4
    else tmp
in y
```
If you build it they will come

As soon as *dice* was put online people started using it in surprising ways we had not foreseen.

Probabilistic Model Checking

Quantum Simulation

Prism

dice

quantum circuit

probabilistic circuit
In both cases, **dice** outperforms existing specialized methods on important examples!

Competitive with well-known simulators like [qsim](https://github.com/Google/qsim) and [qtorch](https://github.com/Google/qtorch) [FSC+ PloS one ‘18]!

---

**Fig. 9.** Scaling plots comparing RUBICON (---), STORM’s symbolic engine (—), and STORM’s explicit engine (—). An “(R)” in the caption denotes random parameters.
Computational Abstractions

Let us think of probability as something that is computed.

Abstraction = Structure of Computation

Two levels of abstraction:

- Probabilistic Programs: “High-level code”
- Probabilistic Circuits: “Machine code”
Thanks

This was the work of many wonderful students/postdoc/collaborators!