Computational Abstractions of Probability Distributions

Guy Van den Broeck

PGM - Sep 24, 2020
Relational Bayesian Networks

Manfred Jaeger
Computer Science Department,
Stanford University, Stanford CA 94305
jaeger@robotics.stanford.edu

Abstract

A new method is developed to represent probabilistic relations on multiple random events. Where previously knowledge bases containing probabilistic rules were used for this purpose,

Probabilistic Decision Graphs – Combining Verification and AI Techniques for Probabilistic Inference

MANFRED JAEGER
Aalborg Universitet, Institut for Datalogi
Fredrik Bajers Vej 7, 9220 Aalborg Ø, Denmark
jaeger@cs.auc.dk

Compiling relational Bayesian networks for exact inference

Mark Chavira a,b, Adnan Darwiche a,
Manfred Jaeger b

a Computer Science Department, UCLA, Los Angeles, CA 90095, United States
b Institut for Datalogi, Aalborg Universitet, Fredrik Bajers Vej 7, DK-9220 Aalborg Ø, Denmark

Available online 17 November 2003
Let me be provocative

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

[VdB KRR15]
Let me be provocative

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

\[
\begin{align*}
3.14 \quad \text{Smokes}(x) \land \\
\text{Friends}(x, y) \\
\Rightarrow \text{Smokes}(y)
\end{align*}
\]
Let me be provocative

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]
Let me be even more provocative

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

We may have gotten stuck in a local optimum?

- Exact probabilistic inference still independence-based
  - Huge effort to extract more local structure from individual tables
- What do you mean, compute probabilities exactly?
  - Statistician: inference = Hamiltonian Monte Carlo
  - Machine learner: inference = variational
- Variable-level causality
Let me be provocative

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

The choice of representing a distribution primarily by its variable-level (in)dependencies is a little arbitrary…

What if we made some different choices?
Computational Abstractions

Let us think of distributions as objects that are computed.

Abstraction = Structure of Computation

‘closer to the metal’

Two examples:
- Probabilistic Circuits
- Probabilistic Programs
Probabilistic Circuits
The Alphabet Soup of probabilistic models
Intractable and tractable models
tractability is a spectrum
Expressive models without compromises
"Every keynote needs a joke and a literature overview slide, not necessarily distinct"
- after Ron Graham
a *unifying framework* for tractable models
Input nodes are tractable (simple) distributions, e.g., indicator functions $p_n(X=1) = [X=1]$
Product nodes are factorizations $\prod_{c \in \text{in}(n)} p_c(x)$
Sum nodes are mixture models \( \sum_{c \in \text{in}(n)} \theta_{n,c} p_c(x) \)
If $p(x) = \sum_i w_i p_i(x)$, (smoothness):

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

$\implies$ integrals are "pushed down" to children

[Darwiche & Marquis JAIR 2001, Poon & Domingos UAI11]
Smoothness + decomposability = tractable MAR

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

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

\[\Rightarrow\] integrals decompose into easier ones
Smoothness + decomposability = tractable MAR

Forward pass evaluation for MAR

\[ \Rightarrow \text{linear in circuit size!} \]

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

- Leafs over \( X_1 \) and \( X_3 \) output \( Z_i = \int p(x_i) dx_i \)
  \[ \Rightarrow \text{for normalized leaf distributions: } 1.0 \]
- Leafs over \( X_2 \) and \( X_4 \) output \( \text{EVI} \)
- Feedforward evaluation (bottom-up)
<table>
<thead>
<tr>
<th>Property</th>
<th>MAR</th>
<th>CON</th>
<th>MOM</th>
<th>MAP</th>
<th>MMAP</th>
<th>ENT</th>
<th>DIV</th>
<th>EXP</th>
<th>Expected Predictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>smoothness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>decomposability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consistency</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>determinism</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>marginal determinism</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>structured decomposability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>paired str. decomposability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Properties
- **SMO**: Smoothness
- **DEC**: Decomposability
- **CON**: Consistency
- **DET**: Determinism
- **MAR-DET**: Marginal determinism
- **STR-DEC**: Structured decomposability
- **P-STR-DEC**: Paired str. decomposability

### Query Types
- **MAR**: Marginal queries
- **CON**: Conditional queries
- **MOM**: Moments (mean...)
- **MAP**: Maximum a posteriori
- **MMAP**: Marginal MAP
- **ENT**: Entropy
- **DIV**: Divergences (KLD...)
- **EXP**: Expected predictions
<table>
<thead>
<tr>
<th>Model Type</th>
<th>smooth</th>
<th>dec.</th>
<th>det.</th>
<th>str.dec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic Circuits (ACs) [Darwiche 2003]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✗️</td>
</tr>
<tr>
<td>Sum-Product Networks (SPNs) [Poon et al. 2011]</td>
<td>✔️</td>
<td>✔️</td>
<td>✗️</td>
<td>✗️</td>
</tr>
<tr>
<td>Cutset Networks (CNets) [Rahman et al. 2014]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✗️</td>
</tr>
<tr>
<td>Probabilistic Decision Graphs [Jaeger 2004]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>PSDDs [Kisa et al. 2014a]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>AndOrGraphs [Dechter et al. 2007]</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
### How expressive are probabilistic circuits?

**density estimation benchmarks**

<table>
<thead>
<tr>
<th>dataset</th>
<th>best circuit</th>
<th>BN</th>
<th>MADE</th>
<th>VAE</th>
<th>dataset</th>
<th>best circuit</th>
<th>BN</th>
<th>MADE</th>
<th>VAE</th>
</tr>
</thead>
<tbody>
<tr>
<td>nltcs</td>
<td><strong>-5.99</strong></td>
<td>-6.02</td>
<td>-6.04</td>
<td><strong>-5.99</strong></td>
<td>dna</td>
<td><strong>-79.88</strong></td>
<td>-80.65</td>
<td>-82.77</td>
<td>-94.56</td>
</tr>
<tr>
<td>msnbc</td>
<td><strong>-6.04</strong></td>
<td>-6.04</td>
<td>-6.06</td>
<td>-6.09</td>
<td>kosarek</td>
<td><strong>-10.52</strong></td>
<td>-10.83</td>
<td>-</td>
<td>-10.64</td>
</tr>
<tr>
<td>plants</td>
<td><strong>-11.84</strong></td>
<td>12.65</td>
<td>12.32</td>
<td>12.34</td>
<td>book</td>
<td>-33.82</td>
<td>-36.41</td>
<td>-33.95</td>
<td><strong>-33.19</strong></td>
</tr>
<tr>
<td>audio</td>
<td>-39.39</td>
<td>-40.50</td>
<td>-38.95</td>
<td><strong>-38.67</strong></td>
<td>movie</td>
<td>-50.34</td>
<td>-54.37</td>
<td>-48.7</td>
<td><strong>-47.43</strong></td>
</tr>
<tr>
<td>jester</td>
<td>-51.29</td>
<td><strong>-51.07</strong></td>
<td>52.23</td>
<td>51.54</td>
<td>webkb</td>
<td>-149.20</td>
<td>-157.43</td>
<td>-149.59</td>
<td><strong>-146.9</strong></td>
</tr>
<tr>
<td>netflix</td>
<td>-55.71</td>
<td>-57.02</td>
<td>-55.16</td>
<td><strong>-54.73</strong></td>
<td>cr52</td>
<td>-81.87</td>
<td>-87.56</td>
<td>-82.80</td>
<td><strong>-81.33</strong></td>
</tr>
<tr>
<td>accidents</td>
<td>-26.89</td>
<td><strong>-26.32</strong></td>
<td>-26.42</td>
<td>-29.11</td>
<td>c20ng</td>
<td>-151.02</td>
<td>-158.95</td>
<td>-153.18</td>
<td><strong>-146.9</strong></td>
</tr>
<tr>
<td>retail</td>
<td><strong>-10.72</strong></td>
<td>-10.87</td>
<td>-10.81</td>
<td>-10.83</td>
<td>bbc</td>
<td><strong>-229.21</strong></td>
<td>-257.86</td>
<td>-242.40</td>
<td>-240.94</td>
</tr>
<tr>
<td>pumbs*</td>
<td>-22.15</td>
<td><strong>-21.72</strong></td>
<td>-22.3</td>
<td>-25.16</td>
<td>ad</td>
<td>-14.00</td>
<td>-18.35</td>
<td><strong>-13.65</strong></td>
<td>-18.81</td>
</tr>
</tbody>
</table>
Want to learn more?

Tutorial (3h)

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

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

Training PCs in Julia with Juice.jl

Training maximum likelihood parameters of probabilistic circuits

```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 ms
```

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

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

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

\[ \Pr_L(X_1 = 1, X_2 = 0, X_3 = 1, X_4 = 0) = \frac{1}{\det(L + I)} \det(L_{1,2}) \]

Computing marginal probabilities is \textit{tractable}.

[Zhang et al. UAI20]
Representing the Determinant as a PC is *not easy*

- Gaussian Elimination
  - Branching and Division

- Laplace Expansion
  - Exponentially many subdeterminants

[Zhang et al. UAI20]
We cannot tractably represent DPPs with classes of PCs

- PSDDs
  - More Tractable
- Deterministic and Decomposable PCs
  - Deterministic PCs with no negative parameters
  - No
  - Decomposable PCs with no negative parameters (SPNs)
  - No
  - Decomposable PCs with negative parameters
    - We don’t know
- Deterministic PCs with negative parameters
  - No
- Fewer Constraints

Stay Tuned!

[Zhang et al. UAI20; Martens & Medabalimi Arxiv15]
The AI Dilemma

Pure Logic

Pure Learning
The AI Dilemma

Pure Logic

- Slow thinking: deliberative, cognitive, model-based, extrapolation
- Amazing achievements until this day
- "Pure logic is brittle"
  noise, uncertainty, incomplete knowledge, …

Pure Learning
The AI Dilemma

Fast thinking: instinctive, perceptive, model-free, interpolation
Amazing achievements recently

“Pure learning is brittle”
- bias, algorithmic fairness, interpretability, explainability, adversarial attacks,
- unknown unknowns, calibration, verification, missing features, missing labels,
- data efficiency, shift in distribution, general robustness and safety
fails to incorporate a sensible model of the world
“Pure learning is brittle”

bias, *algorithmic fairness*, interpretability, *explainability*, adversarial attacks, unknown unknowns, calibration, verification, *missing features*, missing labels, data efficiency, shift in distribution, general robustness and safety

We need to incorporate a sensible probabilistic model of the world
Prediction with Missing Features

<table>
<thead>
<tr>
<th>X^1</th>
<th>X^2</th>
<th>X^3</th>
<th>X^4</th>
<th>X^5</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x8</td>
</tr>
</tbody>
</table>

Train Classifier

Test with missing features

\( x_1 \) \( x_2 \) \( x_3 \) \( x_4 \) \( x_5 \) \( x_6 \)

\( X^1 \) \( X^2 \) \( X^3 \) \( X^4 \) \( X^5 \)

Predict
Expected Predictions

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

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

Generalizes what we’ve been doing all along...

\[
P(C|y) = \sum_m P(C, m|y) = \sum_m P(C|m, y) P(m|y) = \mathbb{E}_{m \sim P(M|y)} P(C|m, y)
\]

\[x^o = \text{observed features} \quad x^m = \text{missing features}\]
Experiments with simple distributions (Naive Bayes) to reason about missing data in logistic regression

“Conformant learning”

[Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]
What about complex classifiers and distributions?

Tractable expected predictions if the classifier is a regression circuit, and the feature distribution is a compatible probabilistic circuits

Recursion that “breaks down” the computation.

For + nodes (n,m), look at subproblems (1,3), (1,4), (2,3), (2,4)

[Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]
Experiments with Probabilistic Circuits

[Khosravi et al. IJCAI19, NeurIPS20, Artemiss20]
```
using ProbabilisticCircuits
pc = load_prob_circuit(zoo_psdd_file("insurance.psdd"));
rc = load_logistic_circuit(zoo_lc_file("insurance.circuit"), 1);

Q8: How different is the insurance costs between smokers and non smokers?

groups = make_observations(["!smoker", ["smoker"]])
exps, _ = Expectation(pc, rc, groups);
println("Smoker : \$ \$(exps[2])\$\n")
println("Non-Smoker: \$ \$(exps[1])\$\n")
println("Difference: \$ \$(exps[2] - exps[1])\$\n")
Smoker : \$ 31355.32630488978
Non-Smoker: \$ 8741.747258310648
Difference: \$ 22613.57904657913
```
What If Training Also Has Missingness

This time we consider decision trees as the classifier

\[
\mathcal{L}(\Theta; D_{\text{train}}) = \frac{1}{|D_{\text{train}}|} \sum_{x^o, y \in D_{\text{train}}} \mathbb{E}_{p_\Phi}(X^m|x^o) \left[ l(y, f_\Theta(x)) \right]
\]

For one decision tree and using MSE loss, can be computed exactly

\[
\theta^*_\ell = \frac{\sum_{x^o, y \in D_{\text{train}}} y \cdot p_\ell(x^o)/p(x^o)}{\sum_{x^o, y \in D_{\text{train}}} p_\ell(x^o)/p(x^o)}
\]

More scenarios such as bagging/boosting in the paper.

[Khosravi et al. IJCAI19, NeurIPS20, Artemiss 20]
Preliminary Experiments

[Image of two graphs showing RMSE vs. missing probability for different methods: XGBoost, Median Impute, Expected Prediction, and a combination of Expected Prediction and ExpLoss + Expected Prediction.]

[Khosravi et al. IJCAI19, NeurIPS20, Artemiss 20]
using ProbabilisticCircuits
pc = load_prob_circuit(zoo_psdd_file("insurance.psdd"));
rc = load_logistic_circuit(zoo_lc_file("insurance.circuit"), 1);

**Q9: 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
Model-Based Algorithmic Fairness: FairPC

Learn classifier given
- features S and X
- training labels D

Fair decision $D_f$ should be independent of the sensitive attribute $S$
Probabilistic Sufficient Explanations

Goal: explain an instance of classification

Choose a subset of features s.t.

1. Given only the explanation it is “probabilistically sufficient”
   
   Under the feature distribution, it is likely to make the prediction to be explained

2. It is minimal and “simple”

[Khosravi et al. IJCAI19, Wang et al. XXAI20]
“Pure learning is brittle”

bias, algorithmic fairness, interpretability, explainability, adversarial attacks, unknown unknowns, calibration, verification, missing features, missing labels, data efficiency, shift in distribution, general robustness and safety

We need to incorporate a sensible probabilistic model of the world
Probabilistic Programs
What are probabilistic programs?

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);

means “flip a coin, and output true with probability \( \frac{1}{2} \)”

Standard (functional) programming constructs: let, if, ...

means “reject this execution if z is not true”
Why Probabilistic Programming?

PPLs are proliferating

Pyro  Edward  HackPPL  Stan  Figaro

Venture, Church, IBAL, WebPPL, Infer.NET, Tensorflow Probability, ProbLog, PRISM, LPADs, CPLologic, 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!
Dice probabilistic programming language

http://dicelang.cs.ucla.edu/

https://github.com/SHoltzen/dice

[Holtzen et al. OOPSLA20 (tentative)]
What is a possible world?

A possible world is a state where all the variables have values.

Let's consider the following code:

```plaintext
let x = flip 0.4 in
let y = flip 0.7 in
let z = x || y in
let x = if z then x else 1 in
in (x, y)
```

<table>
<thead>
<tr>
<th>Execution</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>(1,1)</td>
</tr>
<tr>
<td>Execution B</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>(1,0)</td>
</tr>
<tr>
<td>Execution C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>(0,1)</td>
</tr>
<tr>
<td>Execution D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(1,0)</td>
</tr>
</tbody>
</table>

Probability of each execution:

- P(A) = 0.4 * 0.7
- P(B) = 0.4 * 0.3
- P(C) = 0.6 * 0.7
- P(D) = 0.6 * 0.3
Why should I care? I like PGMs

• Better abstraction:
  • Beyond variable-level dependencies
  • modularity through functions reuse (cf. templative graphical models)
  • intuitive language for local structure; arithmetic
  • data structures
  • first-class observations
First-Class Observations, Functions

```kotlin
fun EncryptChar(key:int, obs:char):Bool {
  let randomChar = ChooseChar() in
  let ciphertext = (randomChar + key) % 26 in
  let _ = observe ciphertext = obs in
  true
  let k = UniformInt(0, 25) in
  let _ = EncryptChar(k, 'H') in ...
  let _ = EncryptChar(k, 'D') in k
}
```

Frequency Analyzer for a Caesar cipher in Dice
What do PGMs bring to the table?

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

Discrete probabilistic programming is the important unsolved open problem!

PGM community knows how to solve this!
Symbolic Compilation to Probabilistic Circuits

- Probabilistic Program
- Symbolic Compilation
- Weighted Boolean Formula
- WMC
- Probabilistic Circuit

Flow:
1. Probabilistic Program
2. Symbolic Compilation
   - Retains Program Structure
3. Weighted Boolean Formula
4. WMC
5. Probabilistic Circuit
   - Logic Circuit (BDD)
   - Circuit compilation
Inference in Dice

(a) Network diagram. (b) Probabilistic program defining the network.

```
fun diamond(s1:Bool):Bool {
    let route = flip1 0.5 in
    let s2 = if route then s1 else F in
    let s3 = if route then F else s1 in
    let drop = flip2 0.0001 in
    s2 ∨ (s3 ∧ ¬drop))
    diamond(diamond(diamond(T)))
```

(c) diamond function. (d) Final BDD.

Network Verification
PPL benchmarks from PL community

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Psi (ms)</th>
<th>DP (ms)</th>
<th>Dice (ms)</th>
<th># Paths</th>
<th>BDD Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grass</td>
<td>167</td>
<td>57</td>
<td>1.0</td>
<td>9.5×10^1</td>
<td>15</td>
</tr>
<tr>
<td>Burglar Alarm</td>
<td>98</td>
<td>10</td>
<td>1.1</td>
<td>2.5×10^2</td>
<td>11</td>
</tr>
<tr>
<td>Coin Bias</td>
<td>94</td>
<td>23</td>
<td>1.0</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>Noisy Or</td>
<td>81</td>
<td>152</td>
<td>1.0</td>
<td>1.6×10^4</td>
<td>35</td>
</tr>
<tr>
<td>Evidence1</td>
<td>48</td>
<td>32</td>
<td>1.0</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>Evidence2</td>
<td>59</td>
<td>28</td>
<td>1.0</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>Murder Mystery</td>
<td>193</td>
<td>75</td>
<td>1.0</td>
<td>1.6×10^1</td>
<td>6</td>
</tr>
</tbody>
</table>
Scalable Inference

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

Time (ms)

# Characters

Length

Length

Length
### Scalable Inference

<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 [48]</td>
<td>772</td>
<td>46</td>
<td>1.0</td>
<td>10</td>
<td>1.1×10^3</td>
<td>28</td>
</tr>
<tr>
<td>Survey [73]</td>
<td>2477</td>
<td>152</td>
<td>2.0</td>
<td>21</td>
<td>1.3×10^4</td>
<td>73</td>
</tr>
<tr>
<td>Alarm [5]</td>
<td>X</td>
<td>X</td>
<td>9.0</td>
<td>509</td>
<td>1.0×10^36</td>
<td>1.3×10^3</td>
</tr>
<tr>
<td>Insurance [7]</td>
<td>X</td>
<td>X</td>
<td>75.0</td>
<td>984</td>
<td>1.2×10^40</td>
<td>1.0×10^5</td>
</tr>
<tr>
<td>Hepar2 [63]</td>
<td>X</td>
<td>X</td>
<td>54.0</td>
<td>1453</td>
<td>2.9×10^69</td>
<td>1.3×10^3</td>
</tr>
<tr>
<td>Hailfinder [1]</td>
<td>X</td>
<td>X</td>
<td>526.0</td>
<td>2656</td>
<td>2.0×10^76</td>
<td>6.5×10^4</td>
</tr>
<tr>
<td>Pigs</td>
<td>X</td>
<td>X</td>
<td>32.0</td>
<td>5618</td>
<td>7.3×10^492</td>
<td>35</td>
</tr>
<tr>
<td>Water [43]</td>
<td>X</td>
<td>X</td>
<td>2926.0</td>
<td>1.0×10^4</td>
<td>3.2×10^54</td>
<td>5.1×10^4</td>
</tr>
<tr>
<td>Munin [3]</td>
<td>X</td>
<td>X</td>
<td>1945.0</td>
<td>8.1×10^5</td>
<td>2.1×10^1622</td>
<td>1.1×10^4</td>
</tr>
</tbody>
</table>
let HYPOVOLEMIA = flip 0.2 in
let LVFAILURE = flip 0.05 in
let STROKEVOLUME =
    if (HYPOVOLEMIA) then
        (if (LVFAILURE) then (discrete(0.98, 0.01, 0.01)) else (discrete(0.50, 0.49, 0.01)))
    else
        (if (LVFAILURE) then (discrete(0.95, 0.04, 0.01)) else (discrete(0.05, 0.90, 0.05)))
in
let LVEDVOLUME =
    if (HYPOVOLEMIA) then
        (if (LVFAILURE) then (discrete(0.95, 0.04, 0.01)) else (discrete(0.01, 0.09, 0.90)))
    else
        (if (LVFAILURE) then (discrete(0.98, 0.01, 0.01)) else (discrete(0.05, 0.90, 0.05)))
    in
...

Alarm Bayesian Network
Why should I care? I like PGMs

• Better abstraction:
  • Beyond variable-level dependencies
  • modularity through functions
    reuse (cf. templative graphical models)
  • intuitive language for local structure; arithmetic
  • data structures
  • first-class observations

• Better inference? correctness? analysis?

import PL.*
Denotational Semantics

- **Goal**: associate with every expression “e” a semantic object.
- **Notation**: semantic bracket: \([\[,\]]\)
  - In Bayesian network: \([[\text{BN}]] = \Pr_{\text{BN}}(.)\)
  - In probabilistic programs: \([[e]](.)\) for all expressions
  - Accepting and distributional semantics:
    \[
    [e]_A \triangleq \sum_v [e] (v), \quad [e]_D (v) \triangleq \frac{1}{[e]_A} [e] (v)
    \]
- **Idea**: don’t need to run ‘flip 0.4’ infinite times to know meaning
Denotational Semantics

+ Formal Inference Rules

\[ [v_1](v) \triangleq (\delta(v_1))(v) \]
\[ [\text{fst}(v_1, v_2)](v) \triangleq (\delta(v_1))(v) \]
\[ [\text{snd}(v_1, v_2)](v) \triangleq (\delta(v_2))(v) \]

\[ [\text{if } v \text{ then } e_1 \text{ else } e_2](v) \triangleq \begin{cases} [e_1](v) & \text{if } v = T \\ [e_2](v) & \text{if } v = F \\ 0 & \text{otherwise} \end{cases} \]

\[ \text{flip } \theta(v) \triangleq \begin{cases} \theta & \text{if } v = T \\ 1 - \theta & \text{if } v = F \\ 0 & \text{otherwise} \end{cases} \]

\[ [\text{observe } v_1](v) \triangleq \begin{cases} 1 & \text{if } v_1 = T \text{ and } v = T, \\ 0 & \text{otherwise} \end{cases} \]

\[ [f(v_1)](v) \triangleq ((T(f))(v_1))(v) \]

\[ [\text{let } x = e_1 \text{ in } e_2](v) \triangleq \sum_{v'} [e_1](v') \times [e_2[x \mapsto v']](v) \]

\[
\begin{align*}
\text{(C-TRUE)}: & \quad T \leadsto (T, T, \emptyset) \\
\text{(C-FALSE)}: & \quad F \leadsto (F, T, \emptyset) \\
\text{(C-IDENT)}: & \quad x \leadsto (x, T, \emptyset) \\
\text{(C-FLIP)}: & \quad \text{fresh } f \\
& \quad \text{flip } \theta \leadsto (f, T, (f \mapsto \theta, T, \overline{\theta} \mapsto 1 - \theta)) \\
\text{(C-OBS)}: & \quad \text{observe } aexp \leadsto (T, \varphi, \emptyset) \\
\text{if } \text{aexp then } e_T \text{ else } e_E \leadsto & \quad \left((\varphi_E \land \varphi_T) \lor ((\overline{\varphi}_E \land \varphi_T), ((\varphi_E \land \varphi_T) \lor ((\overline{\varphi}_E \land \varphi_T), w_T \lor w_E) \right) \\
\text{(C-ITE)}: & \quad e_1 \leadsto (\varphi_1, y, w) \\
& \quad e_2 \leadsto (\varphi_2, y, w) \\
\text{(C-LET)}: & \quad \text{let } x = e_1 \text{ in } e_2 \leadsto (\varphi_2[x \mapsto \varphi_1], y_1 \land y_2[x \mapsto \varphi_1], w_1 \lor w_2) \end{align*}
\]
Provably Correct Inference!
Better Inference?

Exploit modularity

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.

Quick preview:

3. Flip hoisting optimization
4. Eager compilation
From programs to circuits directly:

```latex
let z = flip_1 0.5 in
let x = if z then flip_2 0.6 else flip_3 0.7 in
let y = if z then flip_4 0.7 else x in (x, y)
```

(a) Context-specific independence.

```latex
fun foo(a:Bool, b:Bool, c:Bool):Bool {
  a ∨ b ∨ c
}
```

(c) Structure without independence.
## Compiler Optimizations (sneak preview)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Naive compilation</th>
<th>determinism</th>
<th>flip hoisting + determinism</th>
<th>Eager + flip lifting</th>
<th>Ace baseline</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm</td>
<td>156</td>
<td>140</td>
<td>83</td>
<td>69</td>
<td>422</td>
</tr>
<tr>
<td>water</td>
<td>56,267</td>
<td>65,975</td>
<td>1509</td>
<td>941</td>
<td>605</td>
</tr>
<tr>
<td>insurance</td>
<td>140</td>
<td>100</td>
<td>100</td>
<td>128</td>
<td>492</td>
</tr>
<tr>
<td>hepar2</td>
<td>95</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>495</td>
</tr>
<tr>
<td>pigs</td>
<td>3,772</td>
<td>2490</td>
<td>2112</td>
<td>186</td>
<td>985</td>
</tr>
<tr>
<td>munin</td>
<td>&gt;1,000,000</td>
<td>&gt;1,000,000</td>
<td>109,687</td>
<td>16,536</td>
<td>3,500</td>
</tr>
</tbody>
</table>

Inference time in milliseconds
Conclusions

- Are we already in the age of computational abstractions?
- Probabilistic circuits for learning deep tractable probabilistic models
- Probabilistic programs as the new probabilistic knowledge representation language
Thanks