Recent Advances in Discrete Probabilistic Program Inference

Guy Van den Broeck

VeriProP 2021 - Jul 19, 2021
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 Smokes(x) ∧ Friends(x,y) ⇒ 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

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

[Tehrani et al. PGM20]
Let us think of probability as something that is computed.

Abstraction = Structure of Computation

Two levels of abstraction:

- Probabilistic Programs
- Probabilistic Circuits

“High-level code”
“Machine code”
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!

Inside the Self-Driving Tesla Fatal Accident

By ANJALI SINGHVI and KARL RUSSELL  UPDATED July 12, 2016

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)
```

- `flip 0.5` means “flip a coin, and output true with probability $\frac{1}{2}$”
- `||` and `my_func(x,y)`
- `if z then` 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, CPLogic, 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.

Does not support if-statements!

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.
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(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)))

This doesn't show all the language features of Dice:

- Integers
- Tuples
- Bounded recursion
- Bayesian conditioning
- ...

ECMP equal-cost path protocol: choose randomly which router to forward to
Main routine, combines the networks
0.99 x 0.91 x 0.5 x 0.88 x 0.93 x 0.5 x 0.88 x 0.93
+ 0.99 x 0.91 x 0.5 x 0.19 x 0.33 x 0.5 x 0.88 x 0.93
+ ...
Probabilistic Program Inference

Path enumeration: find all of them!

- Dice
- Dice (Inline)
- Psi
- Psi DP
- WebPPL Exact
- Rejection
Key to Fast Inference: **Factorization** (product nodes)

Easy to see on the graph structure ... how about on the program?

\[
0.99 \times 0.91 \times 0.5 \times 0.88 \times 0.93 \times 0.5 \times 0.88 \times 0.93
+ 0.99 \times 0.91 \times 0.5 \times 0.19 \times 0.33 \times 0.5 \times 0.88 \times 0.93
+ \ldots
\]
Symbolic Compilation in Dice

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

```
let x = flip₁ 0.1 in
let y = if x then flip₂ 0.2 else flip₃ 0.3 in
let z = if y then flip₄ 0.4 else flip₅ 0.5 in z
```

\[
\begin{array}{cccccc}
0.1 & \cdot & 0.2 & \cdot & 0.4 & + & 0.1 & \cdot & 0.8 & \cdot & 0.5 & + & 0.9 & \cdot & 0.3 & \cdot & 0.4 & + & 0.9 & \cdot & 0.7 & \cdot & 0.5
\end{array}
\]

\[
\begin{array}{cccc}
x=T & & y=T & & z=T & & x=T & & y=F & & z=T & & x=F & & y=T & & z=T & & x=F & & y=F & & z=T
\end{array}
\]

\[
f₁ \bar{f₂} f₄ \lor f₁ \bar{f₂} f₅ \lor \bar{f₁} f₃ f₄ \lor \bar{f₁} \bar{f₃} f₅
\]
Symbolic Compilation in Dice

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

Circuit compilation → Logic Circuit (BDD)
An Equivalent BDD to this Program

fun n1(init: bool) {
  let l1succeed = True in
  let l2succeed = True in
  init && l1succeed && l2succeed
}

fun n2(init: bool) {
  let rC = True in
  if rC then
    init && True && True
  else
    init && flip, 0.19 && flip, 0.33
}

f2
f3
f4
f5
f6
f7
f1

Now, how do we compile this?
Compiling the BDD Modularly

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

First, compile the function n1
fun n1(init: bool) {
    let l1succeed = flip 0.99 in
    let l2succeed = flip 0.91 in
    init && l1succeed && l2succeed
}
n1(flip 0.4)

Then, to call n1, substitute for i
Compiling the BDD Modularly

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

n1(flip 0.4)

Then, to call n1, substitute for i
fun n1(init: bool) {
    let l1succeed = flip 0.99 in
    let l2succeed = flip 0.91 in
    init && l1succeed && l2succeed
}
n1(n1(true))

• Calling itself? Size (and therefore inference cost) grows \textit{linearly}
• Build BDD for whole program by combining sub-programs \textit{modularly}
Denotational Semantics
+ Formal Inference Rules

\[
\begin{align*}
[v_1](v) & \triangleq (\delta(v_1))(v) \\
[fst(v_1, v_2)](v) & \triangleq (\delta(v_1))(v) \\
[snd(v_1, v_2)](v) & \triangleq (\delta(v_2))(v) \\
\text{if } v = \text{then } e_1 \text{ else } e_2 \rangle(v) & \triangleq \\
\begin{cases}
[e_1](v) & \text{if } v = \text{T} \\
[e_2](v) & \text{if } v = \text{F} \\
0 & \text{otherwise}
\end{cases} \\
\text{flip } \theta \rangle(v) & \triangleq \\
\begin{cases}
\theta & \text{if } v = \text{T} \\
1 - \theta & \text{if } v = \text{F} \\
0 & \text{otherwise}
\end{cases} \\
\text{observe } v_1 \rangle(v) & \triangleq \\
\begin{cases}
1 & \text{if } v_1 = \text{T} \text{ and } v = \text{T}, \\
0 & \text{otherwise}
\end{cases} \\
[f(v_1)](v) & \triangleq (T(f))(v_1)(v) \\
\text{let } x = e_1 \text{ in } e_2 \rangle(v) & \triangleq \sum_{v'} [e_1](v') \times [e_2[x \mapsto v']](v)
\end{align*}
\]

\[
\begin{array}{ll}
\text{C-TRUE: } & T \rightsquigarrow (T,T,\emptyset) \\
\text{C-FALSE: } & F \rightsquigarrow (F,T,\emptyset) \\
\text{C-IDENT: } & x \rightsquigarrow (x,T,\emptyset) \\
\text{C-FLIP: } & \text{flip } \theta \rightsquigarrow (T,T,\langle f \mapsto \theta, \theta \mapsto \theta, T \mapsto 1 - \theta \rangle) \\
\text{C-OBS: } & \text{observe } aexp \rightsquigarrow (T,T,\emptyset) \\
\text{C-ITE: } & \text{if aexp then } e_1 \text{ else } e_2 \rightsquigarrow ((\varphi_\land \varphi_T) \lor ((\varphi_\land \varphi_E), ((\varphi_\land \varphi_T) \lor ((\varphi_\land \varphi_E), w_T \cup w_E) \\
\text{C-LET: } & \text{let } x = e_1 \text{ in } e_2 \rightsquigarrow ((\varphi_1, y_1, w_1), (\varphi_2, y_2, w_2)
\end{array}
\]
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 x 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 x 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 x 10^3</td>
<td>1.3 x 10^3</td>
</tr>
<tr>
<td>Insurance</td>
<td>x</td>
<td>x</td>
<td>212</td>
<td>984</td>
<td>1.2 x 10^4</td>
<td>1.0 x 10^5</td>
</tr>
<tr>
<td>Hepar2</td>
<td>x</td>
<td>x</td>
<td>54</td>
<td>48</td>
<td>2.9 x 10^4</td>
<td>1.3 x 10^3</td>
</tr>
<tr>
<td>Hailfinder</td>
<td>x</td>
<td>x</td>
<td>618</td>
<td>2656</td>
<td>2.0 x 10^4</td>
<td>6.5 x 10^4</td>
</tr>
<tr>
<td>Pigs</td>
<td>x</td>
<td>x</td>
<td>72</td>
<td>5618</td>
<td>7.3 x 10^4</td>
<td>35</td>
</tr>
<tr>
<td>Water</td>
<td>x</td>
<td>x</td>
<td>2590</td>
<td>1.0 x 10^4</td>
<td>3.2 x 10^5</td>
<td>5.1 x 10^4</td>
</tr>
<tr>
<td>Munin</td>
<td>x</td>
<td>x</td>
<td>1866</td>
<td>8.1 x 10^5</td>
<td>2.1 x 10^16</td>
<td>1.1 x 10^4</td>
</tr>
</tbody>
</table>

More program paths than atoms in the universe
Symbolic Compilation in Dice to Probabilistic Circuits

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

Circuit compilation

Logic Circuit (BDD)

Tractable representations of probability distributions, learnable from data, mapped to GPU/hardware, with many interesting properties!
Learn more about probabilistic circuits?

Tutorial (3h)

Overview Paper (80p)

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

As soon as *dice* was put online people started using it in surprising ways we had not foreseen.
In both cases, *dice* outperforms existing specialized methods on important examples!

**Probalistic Model Checking**

- Weather Factory
- Weather Factory 2
- Herman-13
- Herman-13 (R)
- Herman-17
- Herman-17 (R)
- Queues

**Quantum Simulation**

- qsim sampling with 1 thread
- qsim sampling with 16 threads
- qtorch sampling with 1 thread
- qtorch sampling with 16 threads

Check out CAV talk video or ask Steven Holtzen, Sebastian Junges, or Marcell Vazquez-Chanlatte.
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:

\[
\text{flip } 0.3 \land \text{flip } 0.3 \neq \text{let tmp } = \text{flip } 0.3 \text{ in tmp } \land \text{tmp;}
\]
Compiler Optimization Experiments

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>No Opt</th>
<th>Det</th>
<th>FH</th>
<th>SBK+FH</th>
<th>Ea+FH</th>
<th>Ea+SBK+FH</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALARM</td>
<td>0.15</td>
<td>0.12</td>
<td>0.07</td>
<td>0.10</td>
<td><strong>0.06</strong></td>
<td>0.12</td>
</tr>
<tr>
<td>ANDES</td>
<td>56.73</td>
<td>32.48</td>
<td>3.44</td>
<td>3.42</td>
<td>12.57</td>
<td>7.47</td>
</tr>
<tr>
<td>ASIA</td>
<td>0.06</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
<td><strong>0.02</strong></td>
<td>0.02</td>
</tr>
<tr>
<td>BARLEY</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CANCER</td>
<td>0.04</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
<td><strong>0.02</strong></td>
<td>0.02</td>
</tr>
<tr>
<td>CHILD</td>
<td>0.05</td>
<td>0.04</td>
<td>0.04</td>
<td><strong>0.03</strong></td>
<td>0.03</td>
<td>0.03</td>
</tr>
<tr>
<td>DIABETES</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>EARTHQUAKE</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
<td><strong>0.02</strong></td>
<td>0.02</td>
</tr>
<tr>
<td>HAILFINDER</td>
<td>1.67</td>
<td>0.45</td>
<td>0.53</td>
<td>0.49</td>
<td>0.49</td>
<td>0.41</td>
</tr>
<tr>
<td>HEPAR2</td>
<td>0.13</td>
<td><strong>0.07</strong></td>
<td><strong>0.07</strong></td>
<td>0.12</td>
<td>0.08</td>
<td>0.10</td>
</tr>
<tr>
<td>INSURANCE</td>
<td>0.17</td>
<td>0.08</td>
<td><strong>0.07</strong></td>
<td>0.14</td>
<td>0.16</td>
<td>0.13</td>
</tr>
<tr>
<td>LINK</td>
<td>-</td>
<td>263.38</td>
<td>264.32</td>
<td>265.53</td>
<td>78.75</td>
<td><strong>78.10</strong></td>
</tr>
<tr>
<td>MILDEW</td>
<td>-</td>
<td>71.80</td>
<td>47.01</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MUNIN</td>
<td>-</td>
<td>49.73</td>
<td>22.95</td>
<td>16.34</td>
<td>8.23</td>
<td>3.67</td>
</tr>
<tr>
<td>MUNIN1</td>
<td>-</td>
<td>248.78</td>
<td>196.96</td>
<td>85.39</td>
<td>21.26</td>
<td><strong>10.45</strong></td>
</tr>
<tr>
<td>MUNIN2</td>
<td>-</td>
<td>554.62</td>
<td>431.62</td>
<td>248.68</td>
<td>40.87</td>
<td>23.37</td>
</tr>
<tr>
<td>MUNIN3</td>
<td>-</td>
<td>92.44</td>
<td>67.73</td>
<td>51.72</td>
<td>12.70</td>
<td>7.66</td>
</tr>
<tr>
<td>PATHFINDER</td>
<td>2.28</td>
<td>1.73</td>
<td>2.51</td>
<td>5.20</td>
<td>1.96</td>
<td>4.64</td>
</tr>
<tr>
<td>PIGS</td>
<td>2.33</td>
<td>1.87</td>
<td>1.58</td>
<td>1.54</td>
<td>0.20</td>
<td>0.14</td>
</tr>
<tr>
<td>SACHS</td>
<td>0.04</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>SURVEY</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>WATER</td>
<td>27.04</td>
<td>1.46</td>
<td>1.90</td>
<td>0.87</td>
<td>1.49</td>
<td>0.61</td>
</tr>
<tr>
<td>WIN95PTS</td>
<td>0.09</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
<td>0.03</td>
</tr>
</tbody>
</table>
Conclusions

- Are we already in the age of computational abstractions?
- Probabilistic programs as the new probabilistic knowledge representation language
- Fruitful synthesis of AI and PL/FM
Thanks

This was the work of many wonderful students & collaborators!

References: http://starai.cs.ucla.edu/publications/