Factorized Exact Inference for Discrete Probabilistic Programs

Steven Holtzen, Joe Qian, Todd Millstein, Guy Van den Broeck
UCLA

sholtzen@cs.ucla.edu, qzy@g.ucla.edu, todd@cs.ucla.edu, guyvdb@cs.ucla.edu
Introduction & Motivation

• Our problem: **exact** probabilistic inference for **discrete** programs

Example program

```
x~flip(0.5);
if(x) {
    y~flip(0.4);
} else {
    y~flip(0.6);
}
```

Example inference

\[ \Pr(y) = \frac{1}{2} \]

Why exact inference?

1. No error propagation
2. Core of effective approximation techniques
3. Unaffected by low-probability observations
Introduction & Motivation

• Our problem: **exact** probabilistic inference for **discrete** programs

Example program

\[
x \sim \text{flip}(0.5);
if(x) \{
y \sim \text{flip}(0.4);
\}
\text{else } \{
y \sim \text{flip}(0.6);
\}
\]

Example inference

\[
\Pr(y) = \frac{1}{2}
\]

**Why discrete?**

1. Program constructs (e.g. `if`-statements)
2. Discrete models (graphs, topic models, ...)
Existing techniques for exact inference

1. Enumerative inference
   - Psi
   - WebPPL
   - FairSquare

2. Graphical model compilation
   - Figaro
   - Infer.NET
   - Factorie
Enumerative inference

- Systematically explore all possible assignments to flips in the program

- Scales exponentially with #flips

Assignment Probability: $0.5 \times 0.4 \times 0.4$
Inadequacy of enumerative inference

- Often, we can do better than enumeration

First compute $\Pr(y) = \frac{1}{2}$

Then, compute $\Pr(z)$ without looking at $x$

- Exploits *independence* of $X$ and $Z$ given $Y$
- Can we do this systematically?
Graphical model compilation

X \sim \text{flip}(0.5)

X \sim 

y \sim \text{flip}(0.4)

y \sim \text{flip}(0.6)

x?

N

Y

x

y

\text{Pr}(y|x)

| x | y | \text{Pr}(y|x) |
|---|---|--------------|
| T | T | 0.4          |
| T | F | 0.6          |
| F | T | 0.6          |
| F | F | 0.4          |
Graphical model compilation

• Graph makes dependencies between variables explicit

<table>
<thead>
<tr>
<th></th>
<th>Pr(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>0.5</td>
</tr>
<tr>
<td>F</td>
<td>0.5</td>
</tr>
</tbody>
</table>

| x | y | Pr(y|x) |
|---|---|---------|
| T | T | 0.4     |
| T | F | 0.6     |
| F | T | 0.4     |
| F | F | 0.6     |

| y | z | Pr(z|y) |
|---|---|---------|
| T | T | 0.4     |
| T | F | 0.6     |
| F | T | 0.6     |
| F | F | 0.4     |

• Specialized graph-based inference methods exploit this
Coarseness of graphical models as an abstraction

- Arbitrary choice of abstraction
  \[ x = a \lor b \lor c \lor d \lor e \lor f; \]
- Tiny program, *huge conditional probability tables*

- Obfuscates useful program structure
- Easy for path-based analysis: just run the program!
Coarseness of graphical models as an abstraction

• Graph is *coarse-grained*: if a dependency *can* exist between two variables, they *must* have an edge in the graph

```plaintext
1  z ~ flip_1(0.5);
2  if (z) {
3      x ~ flip_2(0.6);
4      y ~ flip_3(0.7)
5  } else {
6      x ~ flip_4(0.4);
7      y := x
8  }
```

• Graph says there are *no independences*
  • However, program says x and y are indep. *given* z = T
  • Challenging for both graph-based and enumeration inference
# Techniques for exact inference

<table>
<thead>
<tr>
<th>Exploits independence to decompose inference?</th>
<th>Graphical Model Compilation</th>
<th>Symbolic compilation (This work)</th>
<th>Enumeration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Keeps program structure?
Our contribution

• Exact inference for a Boolean-valued loop-free PPL with arbitrary observations
  • Exploits independence, is competitive with graphical model compilation
  • Retains nuanced program structure

• Give semantics for our language, prove our inference correct
Symbolic compilation
Background: Symbolic model checking

- Non-probabilistic programs can be interpreted as *logical formulae* which relate input and output states

\[
\begin{align*}
x & := y; \\
\varphi & = (x' \Leftrightarrow y) \land (y' \Leftrightarrow y) \\
SAT(\varphi \land x' \land y) & = T \\
SAT(\varphi \land x' \land \overline{y}) & = F
\end{align*}
\]
Inference via Weighted Model Counting

- Probabilistic Program
- Symbolic Compilation
- Weighted Boolean Formula
  - WMC
  - Query Result

- Retains Program Structure
- Exploits Independence
- Binary Decision Diagram
Inference via Weighted Model Counting

x := flip(0.5);

<table>
<thead>
<tr>
<th>l</th>
<th>w(l)</th>
</tr>
</thead>
<tbody>
<tr>
<td>f₁</td>
<td>0.4</td>
</tr>
<tr>
<td>f₁</td>
<td>0.6</td>
</tr>
</tbody>
</table>

WMC(φ, w) = \sum_{m \models φ} \prod_{l \in m} w(l).

WMC((x' \iff f₁) \land x \land x', w)?

- A single model: m = x' \land x \land f₁
- w(x') \ast w(x) \ast w(f₁) = 0.4
Symbolic compilation: Flip

- Compositional process $\mathcal{S} \rightsquigarrow (\varphi, w)$

\[
\begin{align*}
\text{fresh } f \\
\text{All variables in the program except for } x \text{ are not changed by this statement}
\end{align*}
\]
Symbolic compilation: Assignment

• Compositional process \( \mathcal{S} \sim (\varphi, w) \)

\[
x := e \sim (x' \leftrightarrow e) \land (\text{rest unchanged}), w
\]

• Captures program structure in the logical expression

\[
x := a \quad || \quad b \quad || \quad c \quad || \quad d \quad || \quad e \quad || \quad f
\]
Symbolic compilation: Sequencing

• Compositional process \( s \sim \rightarrow (\varphi, w) \)

\[
\begin{align*}
    s_1 & \sim \rightarrow (\varphi_1, w_1) & s_2 & \sim \rightarrow (\varphi_2, w_2) \\
    \varphi'_2 & = \varphi_2[x_i \mapsto x'_i, x'_i \mapsto x''_i] \\
    s_1; s_2 & \sim \rightarrow ((\exists x'_i . \varphi_1 \land \varphi'_2)[x''_i \mapsto x'_i], w_1 \uplus w_2)
\end{align*}
\]

• Compile two sub-statements, do some relabeling, then combine them to get the result
Inference via Weighted Model Counting

Probabilistic Program → Symbolic Compilation → Weighted Boolean Formula → WMC → Query Result

- Probabilistic Program
- Symbolic Compilation
- Weighted Boolean Formula
- WMC
- Query Result
- Binary Decision Diagram
Compiling to BDDs

• Consider an example program:
  \[ x \sim \text{flip}(0.4); \]
  \[ y \sim \text{flip}(0.6) \]
  \[
  (x \leftrightarrow f_1) \land (y \leftrightarrow f_2)
  \]

• WMC is efficient for BDDs: \textit{time linear} in size
  • Small BDD = Fast Inference
BDDs exploit conditional independence

- Size of BDD grows linearly with length of Markov chain

```
1  x ~ flip_x(0.5);
2  if(x) { y ~ flip_1(0.6) }
3  else { y ~ flip_2(0.4) };
4  if(y) { z ~ flip_3(0.6) }
5  else { z ~ flip_4(0.9) }
```

Given y=T, does not depend on the value of X: exploits conditional independence
Compiling to BDDs

- BDDs compactly capture complex program structure

\[ x = a \lor b \lor c \lor d \lor e \lor f; \]
Experiments: Well-known Baselines

- Small programs (10s of lines)
Experiments: Markov Chain

- Symbolic (This Work)
- Psi
- WebPPL

Time (s)

Length of Markov Chain
Experiment: Bayesian Network Encodings

• Larger programs (thousands of lines, tens of thousands of flips)

<table>
<thead>
<tr>
<th>Model</th>
<th>Us (s)</th>
<th>BN Time (s)</th>
<th>Size of BDD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarm</td>
<td>1.872</td>
<td>0.21</td>
<td>52k</td>
</tr>
<tr>
<td>Halffinder</td>
<td>12.652</td>
<td>1.37</td>
<td>157k</td>
</tr>
<tr>
<td>Hepar2</td>
<td>7.834</td>
<td>Not reported</td>
<td>139k</td>
</tr>
<tr>
<td>pathfinder</td>
<td>62.034</td>
<td>14.94</td>
<td>392k</td>
</tr>
</tbody>
</table>

Specialized BN inference algorithm

Alarm Network

Pathfinder Network
Probabilistic model checking

• Notable systems: STORM [DE’17], PRISM [KW’11]
• Different family of queries
  • Focus on finding upper/lower bounds on probabilities, not Bayesian inference
• Different symbolic representation of distribution
  • ADDs (aka. MTBDDs) instead of weighted model counting (also used by [CL’13])
  • Cannot exploit independence (but can exploit sparsity)

Inference via WMC

- Has been applied to models other than discrete probabilistic programs


Future Work and Conclusion

• We described a *symbolic exact* approach to inference in discrete probabilistic programs
  • Avoids combinatorial explosion of variable enumeration
  • Systematically exploits nuanced program structure like independence
  • Competitive with exact inference Bayesian network inference techniques
  • Gave a semantics, proved it corresponds with compilation
Future Work and Conclusion

• Extending to more expressive program constructs
  • Loops: symbolic fixpoint construction
  • Procedures: exploiting structure of repeated calls
  • Datatypes: categorical, algebraic types

• Theoretical analysis of inference
  • What program properties make queries harder or easier?

• Alternative symbolic representations beyond BDDs

• Integrating exact discrete inference into systems which do not currently handle it?
Thank you!

Questions?

Contact me: sholtzen@cs.ucla.edu
Extra Slides
Doing better than path-based inference

• Observation: $z$ is independent of $x$ given $y$

Can be summarized by computing

$$\Pr(y) = 0.5 \times 0.4 + 0.5 \times 0.6 = 0.5$$
Doing better than path-based inference

• *Observation:* $z$ is independent of $x$ given $y$

• Program now has only 2 paths
Semantics

• Goal: Prove inference correct
  • Semantics of statements naturally encoded as *conditional probabilities*

\[ x \sim \text{flip}(0.4); \quad (x' \leftrightarrow f_1) \]

<table>
<thead>
<tr>
<th>(x')</th>
<th>(x)</th>
<th>(f_1)</th>
<th>(\text{Pr?})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.6</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.6</td>
</tr>
</tbody>
</table>
Symbolic execution

• SAT queries tell us *reachability*

\[
\varphi = (x' \iff y) \land (y' \iff y)
\]

“Can I start in state \((x \land \neg y)\) and end in state \((x \land y)\)”?

\[
\text{SAT}\left(\varphi \land (x \land \neg y) \land (x' \land y')\right) = F
\]
Transition probability

• Assign a probability to transitioning between states

Problem: This table is huge!

Q: How can we compactly represent it?

| x' | x | f_1 | Pr? 
|----|---|-----|----
| 1  | 1 | 1   | 0.4|
| 1  | 1 | 0   | 0  |
| 1  | 0 | 1   | 0.4|
| 1  | 0 | 0   | 0  |
| 0  | 1 | 1   | 0  |
| 0  | 1 | 0   | 0.6|
| 0  | 0 | 1   | 0  |
| 0  | 0 | 0   | 0.6|

Table shows conditional probability of starting in x and ending in x'
Weighted Model Counting

• Given Boolean formula \( \varphi \), weight function \( w \), \( WMC(\varphi, w) = \sum_{m \models \varphi} \prod_{l \in m} w(l) \).

• WMC queries tell us transition probability

“What is the probability of starting in state \( x \) and ending in state \( x' \)?”

\[
WMC((x' \Leftrightarrow f_1) \land x' \land x, \bar{x}, \bar{f}_1) = 0.4
\]

<table>
<thead>
<tr>
<th>( l )</th>
<th>( w(l) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>1</td>
</tr>
<tr>
<td>( \bar{x} )</td>
<td>1</td>
</tr>
<tr>
<td>( f_1 )</td>
<td>0.4</td>
</tr>
<tr>
<td>( \bar{f}_1 )</td>
<td>0.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( x' )</th>
<th>( x )</th>
<th>( f_1 )</th>
<th>( Pr? )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.6</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.6</td>
</tr>
</tbody>
</table>
Inference via Weighted Model Counting

Probabilistic Program → Symbolic Compilation → Weighted Boolean Formula → WMC → Query Result

Q: How can we do this efficiently? (i.e., without building the whole transition probability table)

\[ x' \leftrightarrow f_1 \]

<table>
<thead>
<tr>
<th>( l )</th>
<th>( w(l) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>1</td>
</tr>
<tr>
<td>( \bar{x} )</td>
<td>1</td>
</tr>
<tr>
<td>( f_1 )</td>
<td>0.4</td>
</tr>
<tr>
<td>( \bar{f}_1 )</td>
<td>0.6</td>
</tr>
</tbody>
</table>

\( x \sim \text{flip}(0.4); \)
Compiling to BDDs

- BDD = compact representation of transition probability table

$x \sim \text{flip}(0.4)$;
$y \sim \text{flip}(0.6)$

$\Pr(x = T, y = T) = 0.4 \times 0.6 \times 1 \times 1$
Querying with BDDs

• Suppose we want to compute \( \text{Pr}(x) \)

\[
x \sim \text{flip}(0.4);
y \sim \text{flip}(0.6)
\]

\[
\text{Pr}(x) = 1.0 \times 0.4 + 0.6 \times 0 = 0.4
\]