Probabilistic Programming

Lecture #1: Introduction

October 24, 2016

Prof. Dr. Ir. Joost-Pieter Katoen
Probabilistic Programming

- What is probabilistic programming?
- What are probabilistic programs good for?
- Why are probabilistic programs intricate?
- What are we going to do in this course?
- What do we expect from you in this course?
Probabilistic Programming

- What is probabilistic programming?
- What are probabilistic programs good for?
- Why are probabilistic programs intricate?
- What are we going to do in this course?
- What do we expect from you in this course?
What is probabilistic programming?

The crux of probabilistic programming is to consider normal-looking programs as if they were probability distributions.

Michael Hicks, Univ. of Maryland
## Probabilistic Programming Languages

<table>
<thead>
<tr>
<th>Probabilistic-C</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figaro</td>
<td>Scala</td>
</tr>
<tr>
<td>Church</td>
<td>Scheme</td>
</tr>
<tr>
<td>Tabular</td>
<td>Windows Excel</td>
</tr>
<tr>
<td>ProbLog</td>
<td>Prolog</td>
</tr>
<tr>
<td>Rely</td>
<td>Guarded Command L</td>
</tr>
</tbody>
</table>
| Venture         | ...........
| PyMC            | Python |
| webPPL          | Javascript |
| ............    | ............ |

[PROBABILISTIC-PROGRAMMING.org](http://PROBABILISTIC-PROGRAMMING.org)
Probabilistic Programming

- What is probabilistic programming?
- What are probabilistic programs good for?
- Why are probabilistic programs intricate?
- What are we going to do in this course?
- What do we expect from you in this course?
Probabilistic Programming: Application Areas

- Quantum Computing
- Security
- Approximate Computing
- Bayesian Networks
- Randomised Algorithms
- Robotics
Probabilistic Programming: Application Areas

- Quantum Computing
- Security
- Approximate Computing
- Bayesian Networks
- Randomised Algorithms
- Robotics
How Statisticians Found Air France Flight 447 Two Years After It Crashed Into Atlantic

MIT Technology Review (2014)
Air France Flight AF 447

Airbus A-330 AF 447

June 1, 2009
AF 447: First search attempts

June 6, 2009

June 7, 2009
AF 447: Where is the wreckage?

East-west cross section Atlantic
A `Probabilistic Program'
The prior distributions

Probability of possible location
Relative to beginning emergency

Reverse drift prior
(ocean and wind drift)
The posterior distributions

Pingers black boxes worked

Pingers of black boxes failed
Bayesian Networks
Rethinking the Bayesian Approach

“In particular, the graphical model formalism that ushered in an era of rapid progress in AI has proven inadequate in the face of [these] new challenges.

A promising new approach that aims to bridge this gap is probabilistic programming, which marries probability theory, statistics and programming languages”

\(^a\)MIT/EECS George M. Sprowls Doctoral Dissertation Award
Probabilistic Programming: Application Areas

Quantum Computing

Security

Approximate Computing

Bayesian Networks

Randomised Algorithms

Robotics
Randomised Algorithms

- A randomised algorithm depends on *random numbers*
  - some decisions are based on random number generations

- That is:
  - Generate a random number $k$ from some range $\{1, \ldots, N\}$
  - Make decisions based on the value of $k$

- Instead of *guessing* whether the order is random, a random order is *imposed*

- Behavior of the algorithm depends on input and on the values produced by the random-number generator
Randomised Algorithms

- **What?** Randomised algorithms depend on random numbers
  - some decisions are based on random number generations

- **Why randomised algorithms?**
  1. Their conceptual simplicity
  2. Their speed: many are faster than deterministic algorithms
     (no particular input elicits worst-case behavior)
  3. Their existence:
     many solve problems that have no deterministic solution

- **Types of randomised algorithms:**
  1. **Las Vegas**: always produces correct results, random run-time
  2. **Monte Carlo**: may produces errors, deterministic run-time
Sample Randomised Algorithms

- Randomised *Quicksort*
- Randomised *Binary Search*
- Rabin-Miller’s *Primality Test* (1980)
- Freivald’s *Matrix Multiplication* (1977)
- Itah-Rodeh’s *Leader Election Protocol* (1990)
Hoare’s Quicksort

Pivot 43

Pivots 15, 67

Pivot 91

Pick a pivot deterministically
Hoare’s Quicksort

Quicksort:

\[
QS(A) \triangleq \\
\text{if } |A| \leq 1 \text{ then return } (A); \\
i := \lfloor |A|/2 \rfloor; \\
A_\less := \{a' \in A \mid a' < A[i]\}; \\
A_\greater := \{a' \in A \mid a' > A[i]\}; \\
\text{return } (QS(A_\less) ++ A[i] ++ QS(A_\greater))
\]

Worst case complexity:
\(O(n^2)\) comparisons
Randomised Quicksort

Pick a pivot randomly
Randomised Quicksort:

\[ rQS(A) \triangleq \]
\[
\text{if } (|A| \leq 1) \text{ then return } (A); \\
i := \text{rand}[1 \ldots |A|]; \\
A_\prec := \{ a' \in A \mid a' < A[i] \}; \\
A_\succ := \{ a' \in A \mid a' > A[i] \}; \\
\text{return } (QS(A_\prec) \leftrightarrow A[i] \leftrightarrow QS(A_\succ))
\]

Worst case complexity:
\[ O(n \log(n)) \text{ expected comparisons} \]
Why: better efficiency!

<table>
<thead>
<tr>
<th>input size</th>
<th>Hoare’s quicksort</th>
<th>randomised quicksort</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>$n^2/4$</td>
<td>$n \log n$</td>
</tr>
<tr>
<td>10</td>
<td>25</td>
<td>33</td>
</tr>
<tr>
<td>100</td>
<td>2,500</td>
<td>664</td>
</tr>
<tr>
<td>1,000</td>
<td>250,000</td>
<td>9,965</td>
</tr>
<tr>
<td>10,000</td>
<td>25,000,000</td>
<td>132,877</td>
</tr>
<tr>
<td>100,000</td>
<td>2,500,000,000</td>
<td>1,660,960</td>
</tr>
</tbody>
</table>

26
Randomized versus Hoare’s quicksort
Sherwood Algorithms

- Basic idea: randomise to **lower** the worst case complexity and **increase** the best case complexity.

- Like Robin Hood in Sherwood Forest, this approach **gives to the poor** (worst case) and **robs from the rich** (best case).

- E.g.: *randomised quicksort* and *randomised binary search*
Matrix Multiplication

Input: three \( O(N^2) \) square matrices \( A, B, \) and \( C \)
Output: yes, if \( A \times B = C \); no, otherwise

Time complexity:
- until end 1960s: cubic
  - 1969: 2.808
  - 1978: 2.796
  - 1979: 2.78
  - 1981: 2.522
  - 1982: 2.527
  - 1984: 2.496
  - 1986: 2.479
  - 1989: 2.376
  - 2014: 2.373
Freivald’s Matrix Multiplication (1977)

Input: three $\mathcal{O}(N^2)$ square matrices $A$, $B$, and $C$
Output: yes, if $A \times B = C$; no, otherwise

Deterministic: compute $A \times B$ and compare with $C$
Complexity: in $\mathcal{O}(N^3)$, best known complexity $\mathcal{O}(N^{2.37})$

Randomised:
1. take a random bit-vector $\bar{x}$ of size $N$
2. compute $A \times (B \bar{x}) - C \bar{x}$
3. output yes if this yields the null vector; no otherwise
4. repeat these steps $k$ times
Complexity: in $\mathcal{O}(k \cdot N^2)$, with false positive with probability $\leq 2^{-k}$
Randomised Leader Election

- A unidirectional ring network of $N$ stations
  - Each node proceeds in a lock-step fashion
  - Each time-slot: read message + process it + send message

- Aim: elected a unique designated leader

- Each round starts by each station randomly selecting its id
  - According to $\text{rand}(1, \ldots, K)$ with $K \ll N$

- Stations pass their selected id around the ring

- Leader := station with highest unique id, if present
Randomised Leader Election

probabilistically choose an id from \([1 \ldots K]\)
Randomised Leader Election

send your selected id to your neighbour
Randomised Leader Election

pass the received id, and check uniqueness own id
Randomised Leader Election

pass the received id, and check uniqueness own id
Randomised Leader Election

pass the received id, and check uniqueness own id
End of First Election Round

no unique leader has been elected
New Election Round

new round and new chances!
Probability to End Elections

Probability leader elected within $L$ rounds ($K=2$)

Probability leader elected within $L$ rounds ($K=4$)
Probabilistic Programming: Application Areas

Quantum Computing

Security

Approximate Computing

Bayesian Networks

Randomised Algorithms

Robotics

40
Security

“Goldwasser and Micali proved (1982) that encryption schemes must be random rather than deterministic [...] an insight that revolutionised the study of encryption and laid the foundation for the theory of cryptographic security.”

used in almost all communication protocols, Internet transactions and cloud computing
The Famous RSA-OAEP Protocol

Oracle \( \text{Enc}_{pk}(m) \):
\[
\begin{align*}
& r \leftarrow \{0, 1\}^{k_0}; \\
& s \leftarrow G(r) \oplus (m \parallel 0^{k_1}); \\
& t \leftarrow H(s) \oplus r; \\
& \text{return } f_{pk}(s \parallel t)
\end{align*}
\]

Oracle \( \text{Dec}_{sk}(c) \):
\[
\begin{align*}
& (s, t) \leftarrow f^{-1}_{sk}(c); \\
& r \leftarrow t \oplus H(s); \\
& \text{if } [s \oplus G(r)]_{k_1} = 0^{k_1} \text{ then return } [s \oplus G(r)]^n \\
& \text{else return } \perp
\end{align*}
\]

Oracle \( G(x) \):
\[
\begin{align*}
& \text{if } x \notin \text{dom}(L_G) \text{ then } L_G[x] \leftarrow \{0, 1\}^{n+k_1}; \text{ return } L_G[x]
\end{align*}
\]

Oracle \( H(x) \):
\[
\begin{align*}
& \text{if } x \notin \text{dom}(L_H) \text{ then } L_H[x] \leftarrow \{0, 1\}^{k_0}; \text{ return } L_H[x]
\end{align*}
\]

Game IND-CCA2:
\[
\begin{align*}
& (sk, pk) \leftarrow \mathcal{KG}(); \\
& (m_0, m_1, \sigma) \leftarrow \mathcal{A}_1(pk); \\
& b \leftarrow \{0, 1\}; \\
& c^* \leftarrow \text{Enc}(pk, m_b); \\
& b' \leftarrow \mathcal{A}_2(pk, c^*, \sigma); \\
& \text{return } b = b'
\end{align*}
\]

Game POW:
\[
\begin{align*}
& (sk, pk) \leftarrow \mathcal{KG}(); \\
& y \leftarrow \{0, 1\}^{n+k_1}; \\
& z \leftarrow \{0, 1\}^{k_0}; \\
& y' \leftarrow I(f_{pk}(y \parallel z)); \\
& \text{return } y = y'
\end{align*}
\]
Its Correctness Proof Took Very Long

- **1994** Purported proof of chosen-ciphertext security
- **2001** Proof establishes a weaker security notion, but desired security can be achieved
  1. ...for a modified scheme, or
  2. ...under stronger assumptions
- **2004** Filled gaps in Fujisaki et al. 2001 proof
- **2009** Security definition needs to be clarified
- **2010** Filled gaps and improved bounds from 2004 proof
- **2012** Improved bound from 2010 proof
What is probabilistic programming?

What are probabilistic programs good for?

Why are probabilistic programs intricate?

What are we going to do in this course?

What do we expect from you in this course?
Three Core Issues

1. Program Correctness
2. Termination
3. Run-Time
Issue 1: Program Correctness

- **Traditional programs:**
  - A program is correct with respect to a (formal) specification
    “for input array A, the output array B is sorted and
    contains all elements contained in A”
  - It refers to the **deterministic input-output relation** of a program
    on a given input, always the same output is provided
  - Partial correctness: if an output is produced, it is correct
  - Total correctness: in addition, the program terminates

- **Probabilistic programs:**
  - They do **not always generate the same output**
  - They generate a **probability distribution over possible outputs**
Let us start simple

\[
x := 0 \quad [0.5] \quad x := 1;
\]
\[
y := -1 \quad [0.5] \quad y := 0
\]

This program admits four runs and yields the outcome:

\[
Pr[x=0, y=0] = Pr[x=0, y=-1] = Pr[x=1, y=0] = Pr[x=1, y=-1] = \frac{1}{4}
\]
For $p$ an arbitrary probability:

```cpp
bool c := true;
int i := 0;
while (c) {
    i := i + 1;
    (c := false [p] c := true)
}
```

This models a geometric distribution with parameter $p$.

$$Pr[i = N] = (1-p)^{N-1} \cdot p \quad \text{for } N > 0$$
For which $p$ and $q$ are these two programs equivalent?
Conditioning

\[ P(A|B) = \frac{P(B|A)P(A)}{P(B)} \]
Conditioning

This program blocks two runs as they violate $x+y = 0$. Outcome:

$$Pr[x=0, y=0] = Pr[x=1, y=-1] = \frac{1}{2}$$

Observations thus normalize the probability of the “feasible” program runs.
One fish is contained within the confines of an opaque fishbowl. The fish is equally likely to be a piranha or a goldfish. A sushi lover throws a piranha into the fish bowl alongside the other fish. Then, immediately, before either fish can devour the other, one of the fish is blindly removed from the fishbowl. The fish that has been removed from the bowl turns out to be a piranha. What is the probability that the fish that was originally in the bowl by itself was a piranha?
The Piranha Puzzle Program

\[
\begin{aligned}
  &f_1 := \text{gf} \ [0.5] \ f_1 := \text{pir}; \\
f_2 := \text{pir} ; \\
s := f_1 \ [0.5] \ s := f_2 ; \\
\text{observe} \ (s = \text{pir})
\end{aligned}
\]

What is the probability that the original fish was a piranha?

\[
\mathbb{P}(f_1 = \text{pir} \mid P \ \text{terminates}) = \frac{1.1/2 + 0.1/4}{1 - 1/4} = \frac{1/2}{3/4} = \frac{2}{3}.
\]
Issue 2: Program Termination

- **Traditional programs:**
  - They terminate (on a given/all inputs), or they do not
  - If they terminate, it takes finitely many steps to do so
  - Showing program termination is undecidable (halting problem)

- **Probabilistic programs:**
  - They terminate (or not) with a certain likelihood
  - They may have diverging runs whose likelihood is zero
  - They may take infinitely many steps (on average) to terminate
  - Showing “probability one” termination is “more” undecidable
This program does not always terminate.
It terminates with probability one.
In finite time, on average.
When does it terminate?

```c
bool c := true;
int nrflips := 0;
while (c) {
    nrflips := nrflips + 1;
    (c := false [0.5] c := true);
}
```

Expected runtime (integral over the bars):

The nrflips-th iteration takes place with probability $\frac{1}{2^{nrflips}}$. 
Consider the one-dimensional (symmetric) random walk:

```c
int x := 10;
while (x > 0) {
    (x := x - 1 [0.5] x := x + 1)
}
```

This program almost surely terminates but requires an infinite expected time to do so.
Issue 3: Program Efficiency

- **Traditional programs:**
  - They have a **deterministic, fixed run-time** for a given input
  - Run-times of terminating programs in **sequence** are **compositional**
    - if $P$ and $Q$ terminate in $n$ and $k$ steps, then $P;Q$ halts in $n+k$ steps
  - Analysis techniques: recurrence equations, tree analysis, etc.

- **Probabilistic programs:**
  - Every run-time has a probability; their **run-time is a distribution**
  - Run-times of “probability one” terminating programs in **sequence** are **not compositional**
  - Analysis techniques: involve reasoning about expected values etc.
Consider the two probabilistic programs:

```c
int x := 1;
bool c := true;
while (c) {
    c := false [0.5] c := true;
    x := 2*x
}
```

```c
while (x > 0) {
    x := x - 1
}
```

Finite expected termination time
How long does it take (on average) to run the programs in sequence?

```
int x := 1;
bool c := true;
while (c) {
    c := false [0.5] c := true;
x := 2*x
}
```

Finite expected termination time

```
while (x > 0) {
x := x - 1
}
```

Finite termination time

\[ \infty \]
Probabilistic Programming

- What is probabilistic programming?
- What are probabilistic programs good for?
- Why are probabilistic programs intricate?
- What are we going to do in this course?
- What do we expect from you in this course?
This course is a block course

Two lectures/week in December 2016 and January 2017
- Mondays 16:15-17:45, room 9U10
- Tuesdays 10:15-11:45, room 9U10
- Weeks 49, 50, 51, 2, 3, and 5

One exercise class/week December 2016 – February 2017
- Fridays 14:15-15:45, room 9U10
- Weeks 49, 50, 2, 3, 5, and 6
- Instructors: Federico Olmedo and Christoph Matheja
Probabilistic Programming Material

- Lecture material = the slides + the lectures + the exercises
  web page: moves.rwth-aachen.de/teaching/ws-1617/

- webPPL website (webppl.org) and its accompanying book (on dippl.org):
  Noah Goodman and Andreas Stuhlmüller:
  *The Design and Implementation of Probabilistic Programming Languages*, 2016

- Course is based on (very recent) literature and the book:
  Annabelle McIver and Carroll Morgan:
  *Abstraction, Refinement and Proof for Probabilistic Systems*
Probabilistic Programming Topics

- Probabilistic programming in webPPL
  examples, recursion, plots, conditioning

- The probabilistic guarded command language pGCL
  examples, syntax, semantics (Markov chains), conditioning,
  non-determinism, [recursion]

- Formal reasoning about probabilistic programs
  weakest pre-conditions, loop invariants, post-conditions

- Almost-sure termination

- Run-time analysis of probabilistic programs
Milestones in Program Verification

- **1969**: Program annotations  
  - Tony Hoare

- **1970**: Weakest precondition  
  - Edsger Dijkstra

- **1977**: Modal logic  
  - Amir Pnueli

- **1981**: Model checking  
  - E. Clarke  
  - A. Emerson  
  - J. Sifakis
Milestones in Verifying Probabilistic Programs

- **1979**: Program semantics
- **1983**: Dynamic logic
- **1997**: Weakest precondition
- **2006**: New programming languages

Dexter Kozen

A. McIver
C. Morgan

A. Pfeffer  D. Roy  N. Goodman
Probabilistic Programming

- What is probabilistic programming?
- What are probabilistic programs good for?
- Why are probabilistic programs intricate?
- What are we going to do in this course?
- What do we expect from you in this course?
You are expected to hand in homework exercises
- working in groups of max. two students
- 50% of the points are required for exam qualification
- at least 90% of the point: bonus point for exam
- webPPL programming exercises + (mostly) theory exercises

Dates
- Exercise class: Fridays 14:15-15:45, Weeks 49-50, 2-3, 5-6
- First exercise series available: December 2, 2016
- First exercise hand-in deadline: December 9, 2016
- Written exam: February (week 7) and March 2017 (week 12)

Reward: 4 ECTS
Probabilistic Programming is .......

- Hot
- Exciting
- Tricky
- Efficient
- Very useful