Probabilistic Programming

Lecture #1: Introduction

Joost-Pieter Katoen

Software Modeling and Verification Chair

RWTH AACHEN UNIVERSITY

RWTH Lecture Series on Probabilistic Programming 2018

Overview

Theme of this course

$\boxed{\text{Principles of Probabilistic Programming}}$

Overview

nature Perspective

"There are several reasons why probabilistic programming could prove to be revolutionary for machine intelligence and scientific modelling."[1]

REVIEW

doi:10.1038/nature14541

Probabilistic machine learning and artificial intelligence

Zoubin Ghahramani[1]

[1] Zoubin Ghahramani leads the Cambridge Machine Learning Group, and holds positions at CMU, UCL, and the Alan Turing Institute.

Probabilistic programs

What?

They are programs with random assignments and conditioning

Why?

- Random assignments: to describe randomised algorithms
- Conditioning: to describe stochastic decision making

What is probabilistic programming?

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

The Programming Languages Enthusiast

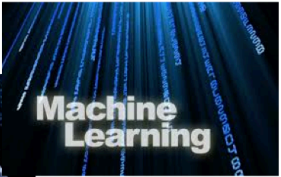
Michael Hicks, Univ. of Maryland

Overview

Applications

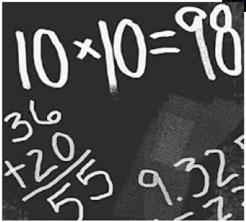
Quantum Computing

Security



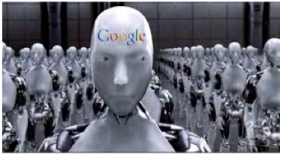
Bayesian Networks

Approximate Computing

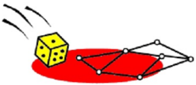


Randomised Algorithms

Robotics



Randomised algorithms



- What? Some decisions are based on coin flips
- Why?
  - Their conceptual simplicity
  - Their speed
    - mostly faster than their deterministic counterpart
    - no particular input elicits worst-case behaviour
  - Their existence
    - many solve problems that have no deterministic solution
- Types
  1. Las Vegas: always produces correct results, random runtime
  2. Monte Carlo: may produce wrong results, deterministic runtime

Famous randomised algorithms

- Randomised Quicksort
- Rabin-Miller's Primality Test (1980)
- Freivald's Matrix Multiplication (1977)
- Lehmann Rabin's Randomised Mutual Exclusion (1981)
- Hermann's Randomised Self-Stabilising Algorithm (1990)
- The Coupon's Collector Algorithm
- . . . . . .

Las Vegas: Sorting by flipping coins

Quicksort:

```
QS(A) =
  if |A| <= 1 { return A; }
  i := ceil(|A|/2);
  A< := {a in A | a < A[i]};
  A> := {a in A | a > A[i]};
  return QS(A<) ++ A[i] ++ QS(A>)
```

Worst case complexity:
$O(N^2)$ comparisons

Randomised Quicksort:

```
rQS(A) =
  if |A| <= 1 { return A; }
  i := Unif[1...|A|];
  A< := {a in A | a < A[i]};
  A> := {a in A | a > A[i]};
  return rQS(A<) ++ A[i] ++ rQS(A>)
```

Worst case complexity:
$O(N \log N)$ expected comparisons

Monte Carlo: Matrix multiplication

Input: three N^2 square matrices A, B, and C

Output: yes, if $A \cdot B = C$; no, otherwise

Time complexity over the years:

- until end 1960s: cubic ($= 3$)
- 1969: 2.808
- 1978: 2.796
- 1979: 2.780
- 1981: 2.522
- 1984: 2.496
- 1989: 2.376
- 2014: 2.373
- 2100: ......

Monte Carlo: Freivald's matrix multiplication

Input: three $\mathcal{O}(N^2)$ square matrices A, B, and C

Output: yes, if $A \times B = C$; no, otherwise

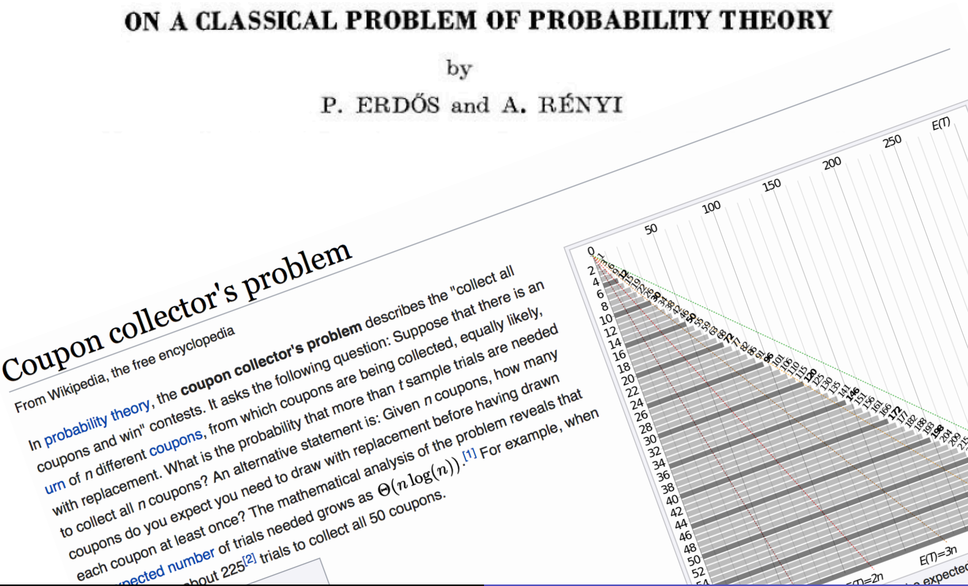
Deterministic: compute $A \times B$ and compare with C

Complexity: in $\mathcal{O}(N^3)$, best known complexity $\mathcal{O}(N^{2.37})$

Randomised:
1. take a random bit-vector $\vec{x}$ of size N
2. compute $A \times (B\,\vec{x}) - C\,\vec{x}$
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 $\leqslant 2^{-k}$

Coupon collector's problem



ON A CLASSICAL PROBLEM OF PROBABILITY THEORY

by

P. ERDŐS and A. RÉNYI

Coupon collector's problem

From Wikipedia, the free encyclopedia

In probability theory, the **coupon collector's problem** describes the "collect all coupons and win" contests. It asks the following question: Suppose that there is an urn of *n* different coupons, from which coupons are being collected, equally likely, with replacement. What is the probability that more than *t* sample trials are needed to collect all *n* coupons? An alternative statement is: Given *n* coupons, how many coupons do you expect you need to draw with replacement before having drawn each coupon at least once? The mathematical analysis of the problem reveals that the ...pected number of trials needed grows as $\Theta(n\log(n))$.[1] For example, when ...bout 225[2] trials to collect all 50 coupons.

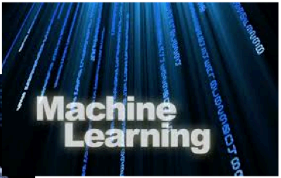
Coupon collector's problem

```
cp := [0,...,0]; // no coupons yet
i := 1; // coupon to be collected next
x := 0: // number of coupons collected
while (x < N) {
   while (cp[i] != 0) {
       i := uniform(1..N) // next coupon
   }
   cp[i] := 1;  // coupon i obtained
   x++;  // one coupon less to go
}
```

Applications

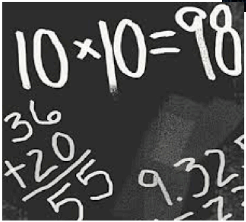
Quantum Computing

Security



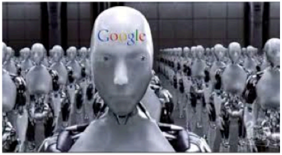
Bayesian Networks

Approximate
Computing



Randomised Algorithms

Robotics



Security

Turing Award Winners 2013

"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 $\mathrm{Enc}_{pk}(m)$:
$r \xleftarrow{\$} \{0,1\}^{k_0}$;
$s \leftarrow G(r) \oplus (m \| 0^{k_1})$;
$t \leftarrow H(s) \oplus r$;
return $f_{pk}(s \| t)$

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

Oracle $G(x)$:
if $x \notin \mathrm{dom}(L_G)$ then $L_G[x] \xleftarrow{\$} \{0,1\}^{n+k_1}$;
return $L_G[x]$

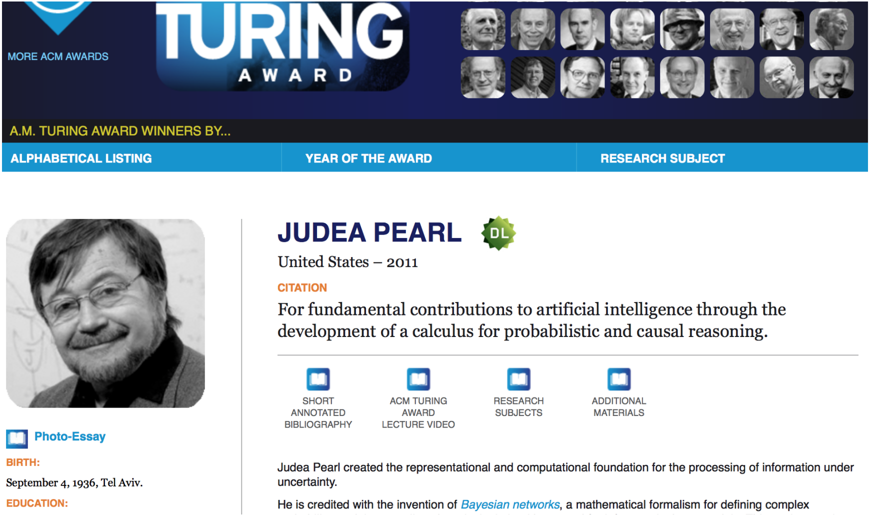
Oracle $H(x)$:
if $x \notin \mathrm{dom}(L_H)$ then $L_H[x] \xleftarrow{\$} \{0,1\}^{k_0}$;
return $L_H[x]$

42

Game IND-CCA2 :
$(sk, pk) \leftarrow \mathcal{KG}(\,)$;
$(m_0, m_1, \sigma) \leftarrow \mathcal{A}_1(pk)$;
$b \xleftarrow{\$} \{0,1\}$;
$c^* \leftarrow \mathrm{Enc}(pk, m_b)$;
$b' \leftarrow \mathcal{A}_2(pk, c^*, \sigma)$;
return $b = b'$

Game POW :
$(sk, pk) \leftarrow \mathcal{KG}()$;
$y \xleftarrow{\$} \{0,1\}^{n+k_1}$;
$z \xleftarrow{\$} \{0,1\}^{k_0}$;
$y' \leftarrow \mathcal{I}(f_{pk}(y \| z))$;
return $y = y'$

The inventor of Bayesian networks



MORE ACM AWARDS

TURING AWARD

A.M. TURING AWARD WINNERS BY...

ALPHABETICAL LISTING | YEAR OF THE AWARD | RESEARCH SUBJECT

Photo-Essay

BIRTH:

September 4, 1936, Tel Aviv.

EDUCATION:

JUDEA PEARL

United States – 2011

CITATION

For fundamental contributions to artificial intelligence through the development of a calculus for probabilistic and causal reasoning.

SHORT ANNOTATED BIBLIOGRAPHY

ACM TURING AWARD LECTURE VIDEO

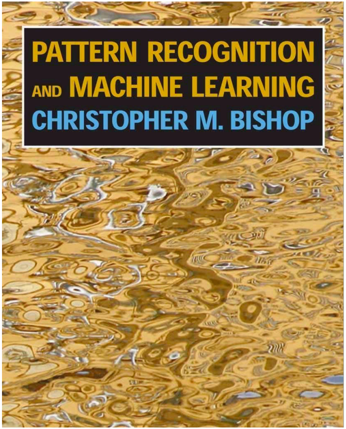
RESEARCH SUBJECTS

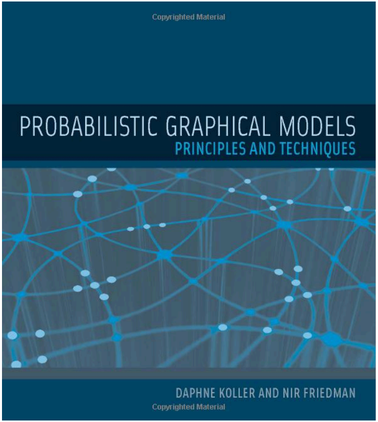
ADDITIONAL MATERIALS

Judea Pearl created the representational and computational foundation for the processing of information under uncertainty.

He is credited with the invention of *Bayesian networks*, a mathematical formalism for defining complex

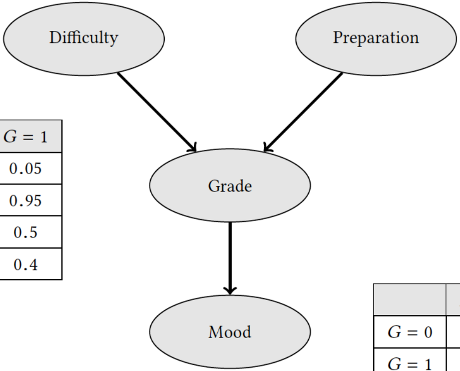
Probabilistic graphical models





Student’s mood after an exam

$D = 0$	$D = 1$
0.6	0.4



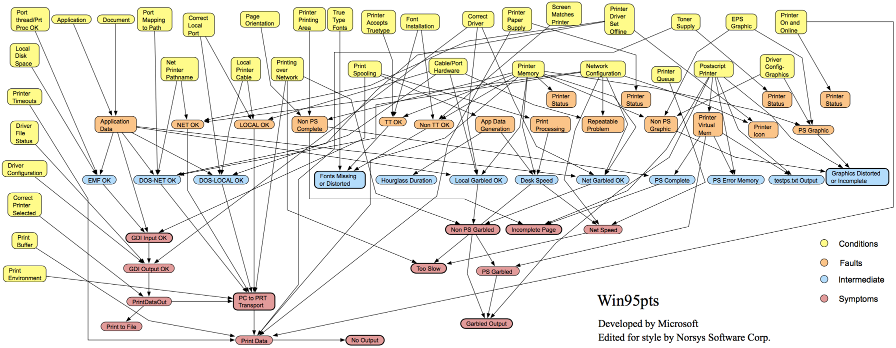
$P = 0$	$P = 1$
0.7	0.3

	$G = 0$	$G = 1$
$D = 0, P = 0$	0.95	0.05
$D = 1, P = 1$	0.05	0.95
$D = 0, P = 1$	0.5	0.5
$D = 1, P = 0$	0.6	0.4

	$M = 0$	$M = 1$
$G = 0$	0.9	0.1
$G = 1$	0.3	0.7

How likely does a student end up with a bad mood after getting a bad grade for an easy exam, **given that** she is well prepared?

Printer troubleshooting in Windows 95



How likely is it that your print is garbled **given that** the ps-file is not and the page orientation is portrait?

[Ramanna *et al.*, Emerging Paradigms in Machine Learning, 2013]

How Statisticians Found Air France Flight 447 Two Years After It Crashed Into Atlantic

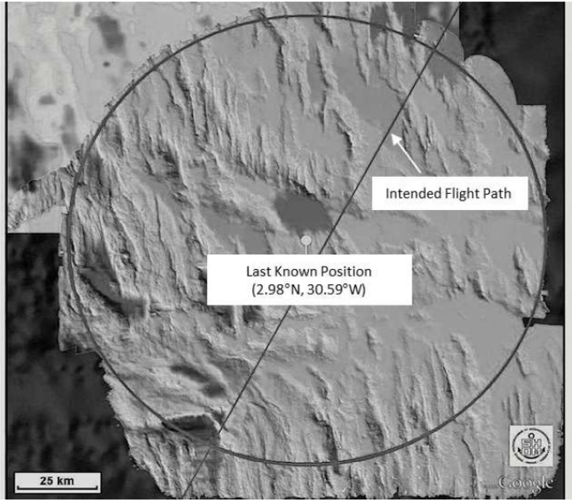
[MIT Technology Review, May 2014]

Air France flight AF-447

Airbus A-330 flight AF-447

June 1, 2009

AF447: Last position



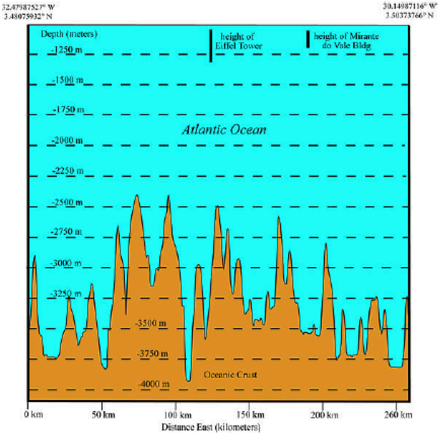
AF447: Failed search attempts

June 6, 2009

June 7, 2009

[Stone *et al.*, Statistical Science, 2013]

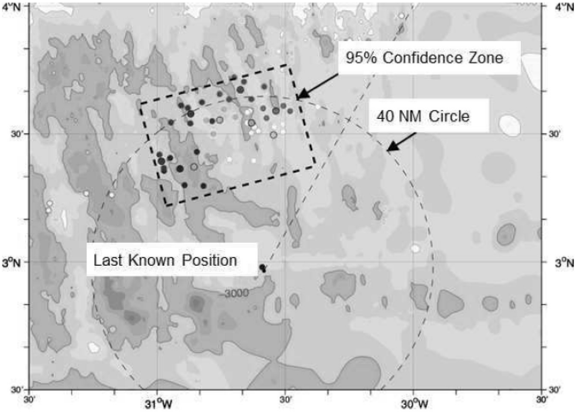
Where is the wreckage?



East-west cross section Atlantic

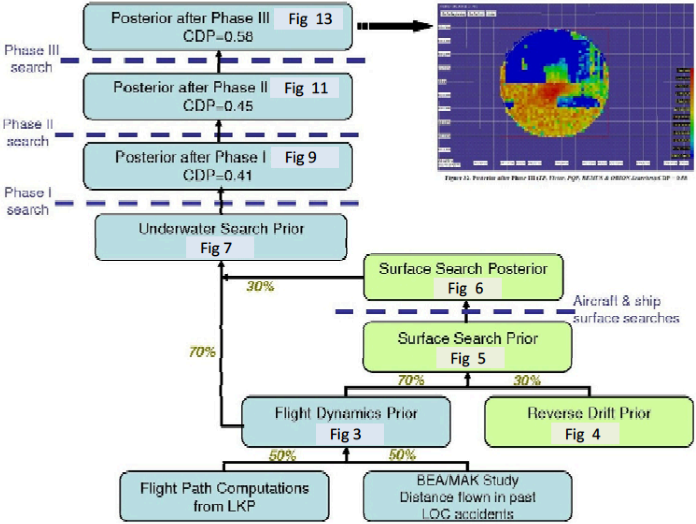
70,000 km^2 were searched, up to 4500 m depth

AF447: Guessed position

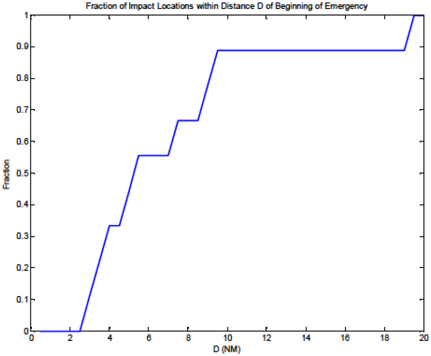


This guided the acoustic search in April 2010.

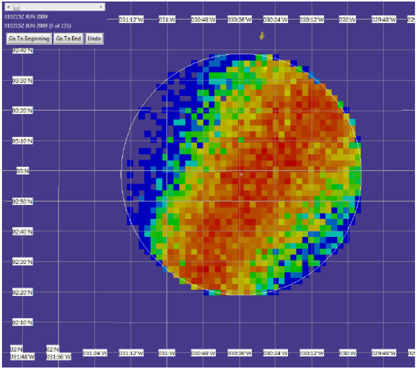
How statisticians came into the play



The priors



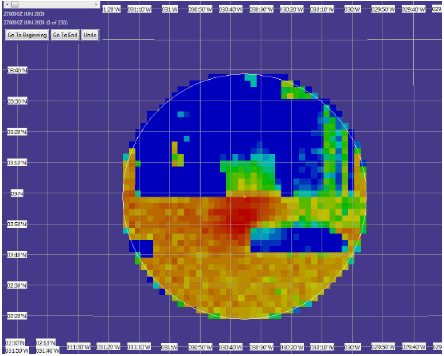
Fraction of impact locations within distance D of beginning of emergency



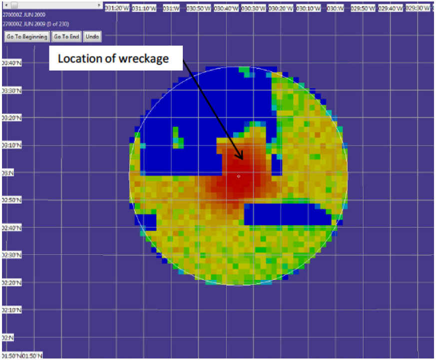
Reverse drift prior (ocean and wind drift) [a]

[a] Currents hard to estimate close to equator.

Two posteriors for location wreckage



Posterior after sonar and UAV search (2010) assuming pingers black boxes function



Posterior pdf assuming pingers of black boxes failed

Rethinking the Bayesian approach

[Daniel Roy, 2011][a]

"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.

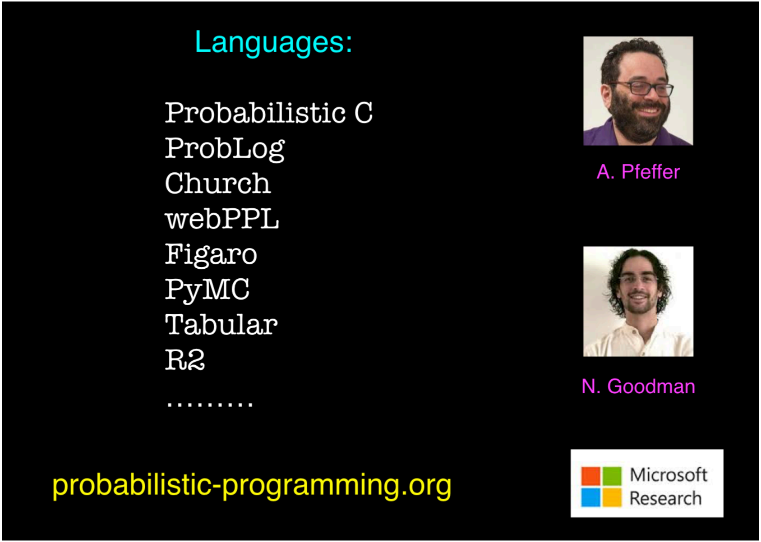
A promising new approach that aims to bridge this gap is probabilistic programming, which marries probability theory, statistics and programming languages"

[a] MIT/EECS George M. Sprowls Doctoral Dissertation Award

Overview

1. What is probabilistic programming?
2. What is probabilistic programming good for?
3. Which probabilistic programming languages do exist?
4. Why are probabilistic programs intricate?
5. What are we going to do in this course?
6. What do we expect from you?

Languages


Languages:
Probabilistic C
ProbLog
Church
webPPL
Figaro
PyMC
Tabular
R2
....
A. Pfeffer
N. Goodman
probabilistic-programming.org
Microsoft
Research

Probabilistic Python

Journal of Statistical Software

July 2010, Volume 35, Issue 4. http://www.jstatsoft.org/

PyMC: Bayesian Stochastic Modelling in **Python**

Anand Patil
University of Oxford

David Huard
McGill University

Christopher J. Fonnesbeck
Vanderbilt University

Probabilistic Scala

Probabilistic Prolog

The Language. Probabilistic Logic Programming

ProbLog makes it easy to express complex, probabilistic models.

```
0.3::stress(X) :- person(X).
0.2::influences(X,Y) :- person(X), person(Y).

smokes(X) :- stress(X).
smokes(X) :- friend(X,Y), influences(Y,X), smokes(Y).

0.4::asthma(X) :- smokes(X).

person(angelika).
person(joris).
person(jonas).
person(dimitar).

friend(joris,jonas).
friend(joris,angelika).
friend(joris,dimitar).
friend(angelika,jonas).
```

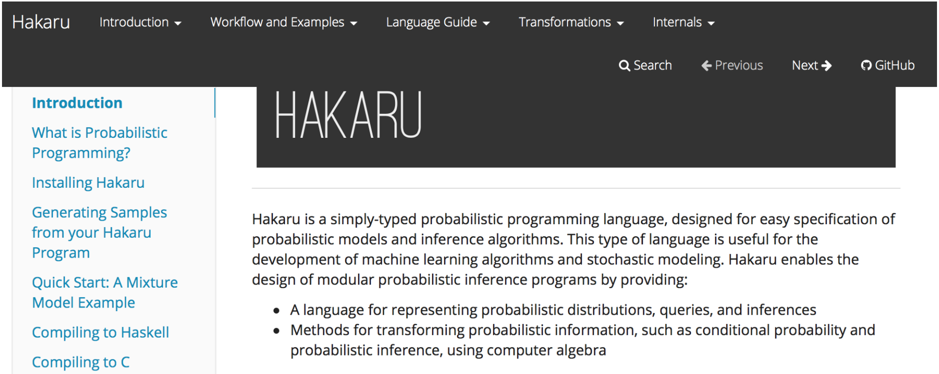
Probabilistic C

Probabilistic programming in C.

Probabilistic generative models can be written in pure C, with only two added keywords: `observe`, to condition on data, and `predict`, to output samples from the program's posterior distribution.

```
#include "probabilistic.h"

int main(int argc, char **argv) {

    double var = 2;
    double mu = normal_rng(1, 5);

    observe(normal_lnp(9, mu, var));
    observe(normal_lnp(8, mu, var));

    predictf("mu %f\n", mu);

    return 0;
}
```

Hakaru


Hakaru Introduction Workflow and Examples Language Guide Transformations Internals

Search Previous Next GitHub

Introduction

What is Probabilistic Programming?

Installing Hakaru

Generating Samples from your Hakaru Program

Quick Start: A Mixture Model Example

Compiling to Haskell

Compiling to C

HAKARU

Hakaru is a simply-typed probabilistic programming language, designed for easy specification of probabilistic models and inference algorithms. This type of language is useful for the development of machine learning algorithms and stochastic modeling. Hakaru enables the design of modular probabilistic inference programs by providing:

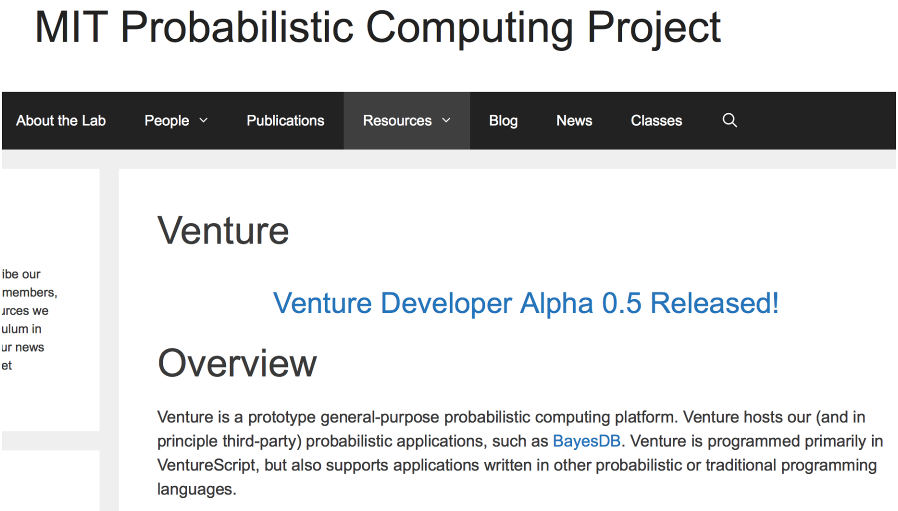
- A language for representing probabilistic distributions, queries, and inferences
- Methods for transforming probabilistic information, such as conditional probability and probabilistic inference, using computer algebra

Hakaru example: Tug-of-war

```
def pulls(strength real):
    normal(strength, 1)

def winner(a real, b real):
    a_pull <~ pulls(a)
    b_pull <~ pulls(b)
    return (a_pull > b_pull)

alice <~ normal(0,1)
bob   <~ normal(0,1)
carol <~ normal(0,1)

match1 <~ winner(alice, bob)
match2 <~ winner(bob, carol)
match3 <~ winner(alice, carol)
```

see: http://hakaru-dev.github.io/intro/probprog/

Venture


MIT Probabilistic Computing Project
About the Lab
People
Publications
Resources
Blog
News
Classes
ibe our
members,
ırces we
ulum in
ur news
et
Venture
Venture Developer Alpha 0.5 Released!
Overview
Venture is a prototype general-purpose probabilistic computing platform. Venture hosts our (and in principle third-party) probabilistic applications, such as BayesDB. Venture is programmed primarily in VentureScript, but also supports applications written in other probabilistic or traditional programming languages.

Scenic

Scenic: Language-Based Scene Generation

Daniel Fremont, Xiangyu Yue, Tommaso Dreossi, Shromona Ghosh, Alberto L. Sangiovanni-Vincentelli and Sanjit A. Seshia

EECS Department
University of California, Berkeley
Technical Report No. UCB/EECS-2018-8
April 18, 2018

http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-8.pdf

Synthetic data has proved increasingly useful in both training and testing machine learning models such as neural networks. The major problem in synthetic data generation is producing meaningful data that is not simply random but reflects properties of real-world data or covers particular cases of interest. In this paper, we show how a probabilistic programming language can be used to guide data synthesis by encoding domain knowledge about what data is useful. Specifically, we focus on data sets arising from scenes, configurations of physical objects: for example, images of cars on a road. We design a domain-specific language, Scenic, for describing scenarios that are distributions over scenes. The syntax of Scenic makes it easy to specify complex relationships between the positions and orientations of objects. As a

Scenic example: a badly parked car

Figure 4: A scene of a badly parked car.

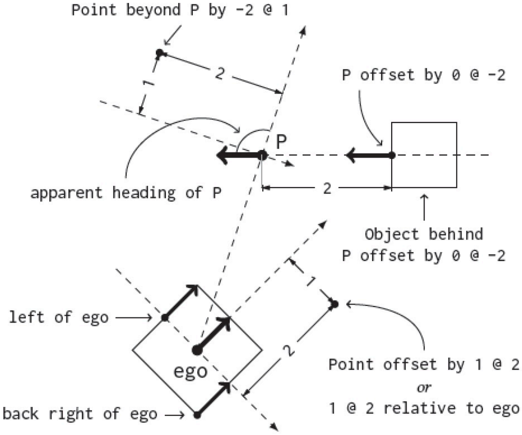
```
spot = OrientedPoint on visible curb
badAngle = Uniform(1.0, -1.0) * (10, 20) deg
Car left of (spot offset by -0.5 @ 0), \
    facing badAngle relative to roadDirection
```

A bit of Scenic's syntax

$$
\begin{aligned}
\textit{scenario} &:= (\texttt{import}\ \textit{file})^*\ \ (\textit{statement})^* \\
\textit{boolean} &:= \texttt{True} \mid \texttt{False} \mid \textit{booleanOperator} \\
\textit{scalar} &:= \textit{number} \mid \textit{distribution} \mid \textit{scalarOperator} \\
\textit{distribution} &:= \textit{baseDist} \mid \texttt{resample}(\textit{distribution}) \\
\textit{vector} &:= \textit{scalar}\ \texttt{@}\ \textit{scalar} \mid \textit{Point} \mid \textit{vectorOperator} \\
\textit{heading} &:= \textit{scalar} \mid \textit{OrientedPoint} \mid \textit{headingOperator} \\
\textit{direction} &:= \textit{heading} \mid \textit{vectorField} \\
\textit{value} &:= \textit{boolean} \mid \textit{scalar} \mid \textit{vector} \mid \textit{direction} \\
&\quad \mid \textit{region} \mid \textit{instance} \mid \textit{instance}\,\texttt{.}\,\textit{property} \\
\textit{classDefn} &:= \texttt{class}\ \textit{class}[(\textit{superclass})]\texttt{:} \\
&\qquad\quad (\textit{property}\texttt{:}\ \ \textit{defaultValue})^* \\
\textit{instance} &:= \textit{class}\ \ \textit{specifier}\texttt{,}\ \dots \\
\textit{specifier} &:= \texttt{with}\ \textit{property}\ \ \textit{value} \mid \textit{posSpec} \mid \textit{headSpec}
\end{aligned}
$$

Syntax	Distribution
`(`*low*`, `*high*`)`	uniform on interval of $\mathbb{R}$
`Uniform(`*value*`, …)`	uniform over given values
`Discrete({`*value*`: `*wt*`, …})`	discrete with given weights
`Normal(`*mean*`, `*stdDev*`)`	normal with given μ, σ

Scenic operators



Example scenes


Edward

DEEP PROBABILISTIC PROGRAMMING

Dustin Tran
Columbia University

Matthew D. Hoffman
Adobe Research

Rif A. Saurous
Google Research

Eugene Brevdo
Google Brain

Kevin Murphy
Google Research

David M. Blei
Columbia University

ABSTRACT

We propose Edward, a Turing-complete probabilistic programming language. Edward defines two compositional representations—random variables and inference.

WebPPL

The **Design** and **Implementation** of **Probabilistic Programming Languages**

The WebPPL language

WebPPL (pronounced 'web people'), is a small probabilistic programming language built on top of a (purely functional) subset of Javascript. The language is intended to be simple to implement, fairly pleasant to write models in, and a good intermediate target for other languages (such as Church). This page documents the language, illustrating with some very simple examples. Further examples are presented in the examples pages.

`webppl.org`

Overview

Three elementary issues

1. Program correctness

2. Termination

3. The runtime of a program

Issue 1: Program correctness

- Classical 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"
  - Defines a deterministic input-output relation
  - 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

Issue 2: Termination

- Classical programs:
  - They terminate (on a given/all inputs), or they do not
  - If they terminate, they take 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

    even if they terminate with probability one!
  - Showing "probability-one" termination is "more" undecidable
    - and showing they do in finite time on average, even more!

Issue 3: The program's runtime

- Classical programs:
  - They have a deterministic, fixed run-time for a given input
  - Runtimes 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 runtime has a probability; their runtime is a distribution
  - Runtimes of "probability-one" terminating programs may not sum up

    if P and Q terminate in n and k steps on average,
    then P;Q may need <u>infinitely</u> many steps on average
  - Analysis techniques: involve reasoning about expected values etc.

Overview

This course's topics (1)

- Probabilistic programming in `webPPL`
  - examples, recursion, plots, conditioning
- The probabilistic guarded command language `pGCL`
  - examples, syntax, semantics (Markov chains), conditioning, [recursion]
- Formal reasoning about probabilistic programs
  - weakest pre-conditions, loop invariants, post-conditions, conditioning

This course's topics (2)

- Almost-sure termination
  - positive a.s.-termination, hardness, stochastic ranking functions
- Analysing runtimes of probabilistic programs
  - examples, finite versus infinite expected runtime, wp-reasoning
- Bayesian networks
  - examples, BNs as programs, BN analysis by program verification

Course material

- Lecture material = the slides + the lectures + recent papers
- webPPL and its accompanying book:
  Noah Goodman and Andreas Stuhlmüller:
  The Design and Implementation of Probabilistic Programming Languages, 2016.
  Available from `dippl.org`
- Course is based on (very recent) papers and the book:
  Annabelle McIver and Carroll Morgan:
  Abstraction, Refinement and Proof for Probabilistic Systems, 2005.
  Available from
  `http://www.cse.unsw.edu.au/~carrollm/arp/ARP1-54.pdf`

Lectures

Lecture

- Tue 14:30–16:00 (5052), Thu 14:30–16:00 (5055)
- Oct 11, 18, **19**, 23, 25, 30
- Nov 6, **9**, 13, 15, 20, 27, 29
- Dec 4, 6, 11, 13, 18, 20
- January 8, 10, 31
- Check regularly course web page for possible "no shows"

Website

`http://moves.rwth-aachen.de/teaching/ws-1819/probabilistic-programming/`

Overview

Exercises

- You are expected to hand in homework exercises
  - typically on an almost weekly basis
  - working in groups of maximally three students
  - practical exercises and theory exercises
  - solutions discussed at the exercise classes
- Start:
  - First exercise series: Fri Oct 19
  - First exercise class: Fri Oct 26
  - The lecture and exercise class are swapped on Nov 8 and 9
- Assistant: **Christoph Matheja**

Examination

- Form: written exam
- Qualification: $\geq 40\%$ of points in exercise series
- Dates:
  - **February 25, 2019** (10:00-12:00)
  - **March 27, 2019** (10:00-12:00)
- All slides may be brought to the exam