Probabilistic Programming

Quantitative Modeling for the Masses?

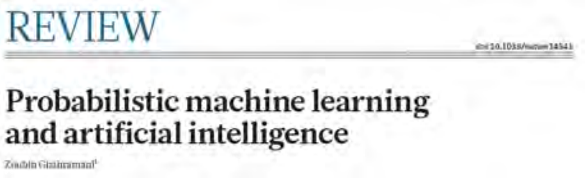
Joost-Pieter Katoen

UNIVERSITY OF TWENTE.

MMB 2018 Conference, Erlangen

nature Perspective

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


REVIEW

Probabilistic machine learning and artificial intelligence

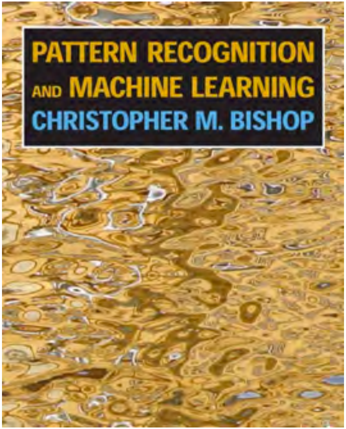
Zoubin Ghahramani[1]

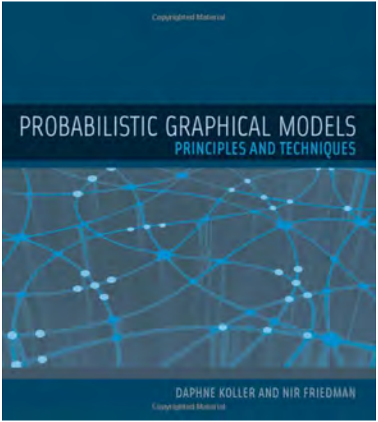
Why? Probabilistic programming

1. … obviates the need to manually provide inference methods
2. … enables rapid prototyping
3. … clearly separates the model and the inference procedures

[1] Ghahramani leads the Cambridge ML Group, and is with CMU, UCL, and Turing Institute.

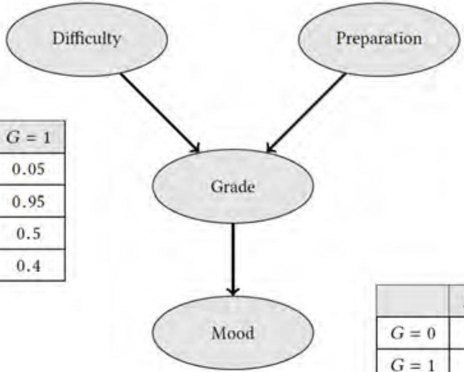
Probabilistic graphical models





Student's mood after an exam

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



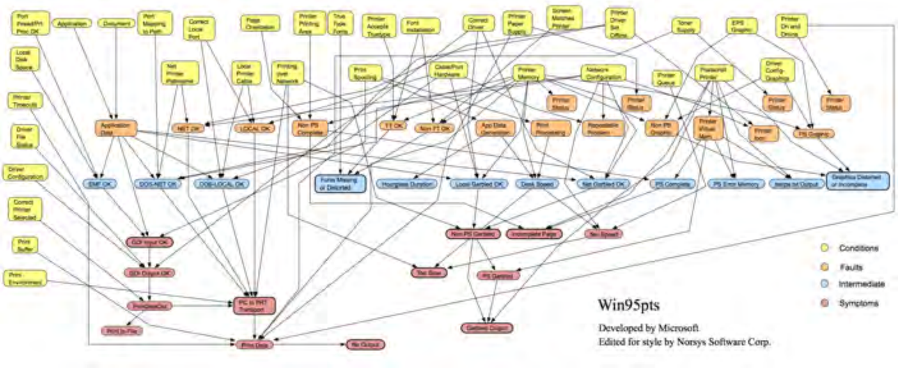
$P = 0$	$P = 1$
0.7	0.3

	$G = 0$	$G = 1$
$D = 0, P = 0$	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?

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

Probabilistic programs

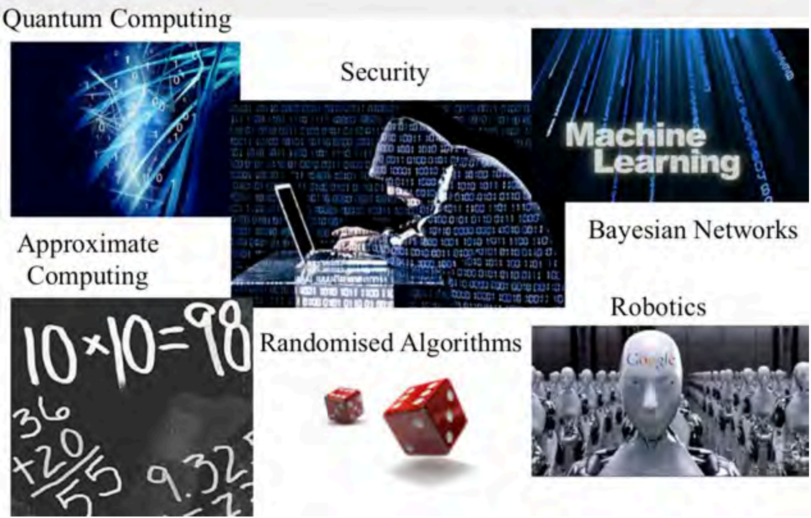
What?

Programs with random assignments and conditioning

Why?

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

Applications



Languages


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

Roadmap

1. An “assembler” probabilistic programming language
2. Bayesian inference by program analysis
3. Termination
4. Runtime analysis
5. How long to sample a Bayes’ network?
6. Epilogue

Overview

Probabilistic GCL

Kozen

McIver

Morgan

- `skip` — empty statement
- `diverge` — divergence
- `x := E` — assignment
- `observe (G)` — **conditioning**
- `prog1 ; prog2` — sequential composition
- `if (G) prog1 else prog2` — choice
- `prog1 [p] prog2` — **probabilistic choice**
- `while (G) prog` — iteration

Let’s start simple

```
x := 0 [0.5] x := 1;
y := 0 [0.5] y := -1;
observe (x+y = 0)
```

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

$$Pr[x = 0, y = 0] = Pr[x = 1, y = -1] = 1/2$$

Observations thus normalize the probability of the “feasible” program runs

A loopy program

For $0 < p < 1$ an arbitrary probability:

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

The feasible program runs have a probability $\sum_{N \geq 0} (1-p)^{2N} \cdot p \; = \; \frac{1}{2-p}$

This program models the distribution:

$$Pr[i = 2N+1] \; = \; (1-p)^{2N} \cdot p \cdot (2-p) \quad \text{for } N \geq 0$$

$$Pr[i = 2N] = 0$$

Or, equivalently

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

This is also known as rejection sampling

Weakest pre-expectations [McIver & Morgan 2004]

An expectation[2] maps states onto $\mathbb{R}_{\geq 0} \cup \{ \infty \}$. It is the quantitative analogue of a predicate. Let $f \leq g$ iff $f(s) \leq g(s)$, for every state s.

An expectation transformer is a total function between two expectations.

The transformer $wp(P, f)$ yields the least expectation e on P's initial state ensuring that P terminates with expectation f.

Annotation $\{e\}\, P\, \{f\}$ holds for total correctness iff $e \;\leq\; wp(P, f)$.

Weakest liberal pre-expectation $wlp(P, f) \;=\;$ "$wp(P, f) + Pr[P \text{ diverges}]$".

[2] ≠ expectations in probability theory.

Expectation transformer semantics of pGCL

Syntax	Semantics $wp(P, f)$
`skip`	f
`diverge`	0
`x := E`	$f(x := E)$
`observe (G)`	$[G] \cdot f$
`P1 ; P2`	$wp(P_1, wp(P_2, f))$
`if (G) P1 else P2`	$[G] \cdot wp(P_1, f) + [\neg G] \cdot wp(P_2, f)$
`P1 [p] P2`	$p \cdot wp(P_1, f) + (1-p) \cdot wp(P_2, f)$
`while (G)P`	$\mu X. ([G] \cdot wp(P, X) + [\neg G] \cdot f)$

μ is the least fixed point operator wrt. the ordering $\leq$.

wlp-semantics differs from wp-semantics only for **while** and **diverge**.

Examples

1. Let program P be:

```
x := 5 [4/5] x := 10
```

   For $f = x$, we have

$$wp(P, x) = \frac{4}{5} \cdot wp(x := 5, x) + \frac{1}{5} \cdot wp(x := 10, x) = \frac{4}{5} \cdot 5 + \frac{1}{5} \cdot 10 = 6$$

2. Let program P' be:

```
x := x+5 [4/5] x := 10
```

   For $f = x$, we have:

$$wp\left(P', x\right) = \frac{4}{5} \cdot wp(x + := 5, x) + \frac{1}{5} \cdot wp(x := 10, x) = \frac{4}{5} \cdot (x+5) + \frac{1}{5} \cdot 10 = \frac{4x}{5} + 6$$

3. For program P' (again) and $f = [x = 10]$, we have:

$$\begin{aligned} wp\left(P', [x{=}10]\right) &= \frac{4}{5} \cdot wp(x := x+5, [x{=}10]) + \frac{1}{5} \cdot wp(x := 10, [x{=}10]) \\ &= \frac{4}{5} \cdot [x+5 = 10] + \frac{1}{5} \cdot [10 = 10] \\ &= \frac{4 \cdot [x{=}5] + 1}{5} \end{aligned}$$

An operational perspective

For program P, input s and expectation f:

$$\frac{wp(P, f)(s)}{wlp(P, \mathbf{1})(s)} \quad = \quad \mathbb{E}\{\ \text{Rew}_s^{[\![P]\!]}(\Diamond sink \mid \neg \Diamond \lightning)\ \}$$

The ratio $wp(P, f) / wlp(P, \mathbf{1})$ for input s equals[3] the conditional expected reward to reach successful terminal state *sink* while satisfying all `observe`'s in MC $[\![P]\!]$.

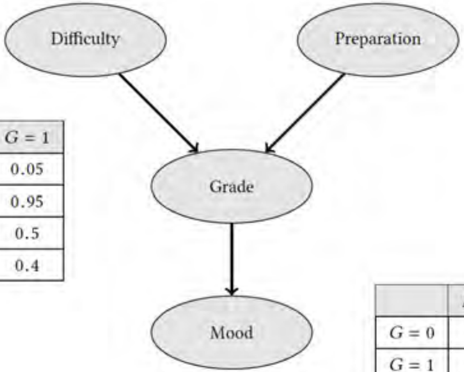
For finite-state programs, wp-reasoning can be done with model checkers such as `PRISM` and `Storm` (`www.stormchecker.org`).

[3] Either both sides are equal or both sides are undefined.

Overview

Bayesian inference

$D = 0$	$D = 1$
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

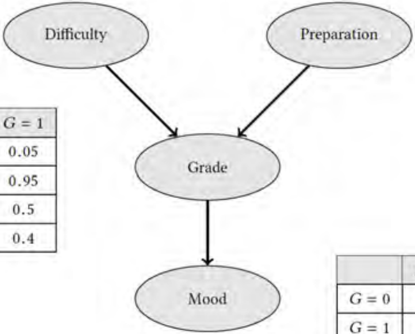
	$M = 0$	$M = 1$
$G = 0$	0.9	0.1
$G = 1$	0.3	0.7

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?

Bayesian inference

$D=0$	$D=1$
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

$$Pr(D=0, G=0, M=0 \mid P=1) = \frac{Pr(D=0, G=0, M=0, P=1)}{Pr(P=1)}$$

$$= \frac{0.6 \cdot 0.5 \cdot 0.9 \cdot 0.3}{0.3} = \mathbf{0.27}$$

Bayesian inference by program verification

- Exact inference of Bayesian networks is NP-hard
- Approximate inference of BNs is NP-hard too
- Typically simulative analyses are employed
  - Rejection Sampling
  - Markov Chain Monte Carlo (MCMC)
  - Metropolis-Hastings
  - Importance Sampling
  - ......
- Here: weakest precondition-reasoning

I.i.d-loops

Loop `while(G)P` is iid wrt. expectation f whenever:

both $wp(P, [G])$ and $wp(P, [\neg G] \cdot f)$ are unaffected by P.

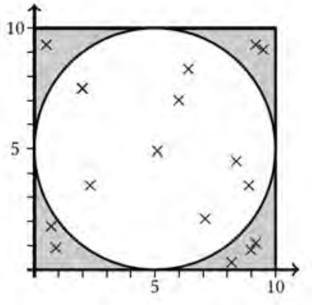
f is *unaffected* by P if none of f's variables are modified by P:

x is a variable of f iff $\exists s. \exists v, u: \quad f(s[x = v]) \neq f(s[x = u])$

If g is unaffected by program P, then: $wp(P, g \cdot f) = g \cdot wp(P, f)$

Example: sampling within a circle

```
while ((x-5)**2 + (y-5)**2 >= 25){
    x := uniform(0..10);
    y := uniform(0..10)
}
```



This program is iid for every f, as both are unaffected by P's body:

$$wp(P, [G]) \quad = \quad \frac{48}{121} \quad \text{and}$$

$$wp(P, [\neg G] \cdot f) \quad = \quad \frac{1}{121} \sum_{i=0}^{10p} \sum_{j=0}^{10p} [(i/p - 5)^2 + (j/p - 5)^2 < 25] \cdot f(x/(i/p), y/(j/p))$$

Weakest precondition of iid-loops

If `while(G)P` is iid for expectation f, it holds for every state s:

$$wp(\texttt{while(G)P}, f)(s) \;=\; [G](s) \cdot \frac{wp(P, [\neg G]\cdot f)(s)}{1 - wp(P, [G])(s)} + [\neg G](s) \cdot f(s)$$

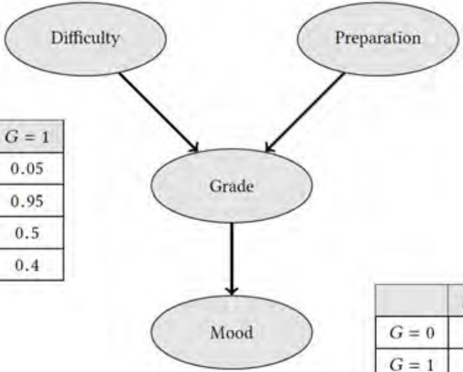
where we let $\frac{0}{0} = 0$.

Proof: use $wp(while_n(G)P, f) = [G] \cdot wp(P, [\neg G]\cdot f) \cdot \sum_{i=0}^{n-2} \left(wp(P, [G])^i\right) + [\neg G] \cdot f$

No loop invariant, martingale, or ranking function needed. Fully automatable.

Bayesian inference

$D=0$	$D=1$
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?

Bayesian networks as programs

- Take a topological sort of the BN's vertices, e.g., D; P; G; M
- Map each conditional probability table (aka: node) to a program, e.g.:

```
if (xD = 0 && xP = 0) {
   xG := 0 [0.95] xG := 1
   } else if (xD = 1 && xP = 1) {
   xG := 0 [0.05] xG := 1
   } else if (xD = 0 && xP = 1) {
   xG := 0 [0.5] xG := 1
   } else if (xD = 1 && xP = 0) {
   xG := 0 [0.6] xG := 1
}
```

	$G = 0$	$G = 1$
$D = 0, P = 0$	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

- Condition on the evidence, e.g., for $P = 1$ we get:

```
repeat { progD ; progP; progG ; progM } until (P=1)
```

Properties of BN programs

```
repeat { progD ; progP; progG ; progM } until (P=1)
```

1. Every BN-program naturally represents rejection sampling
2. Every BN-program is iid for every expectation f
3. Every BN-program almost surely terminates
4. A BN-program's size is linear in the BN's size

Soundness

For BN B over V with evidence *obs* for $O \subseteq V$ and value $\underline{v}$ for node v:

$$\underbrace{wp\left(\texttt{prog}(B, obs), \bigwedge_{v \in V \setminus O} x_v = \underline{v}\right)}_{\text{wp of the BN program of } B} = \underbrace{Pr\left(\bigwedge_{v \in V \setminus O} v = \underline{v} \;\middle|\; \bigwedge_{o \in O} o = obs(o)\right)}_{\text{joint distribution of } B}$$

where $\texttt{prog}(B, obs)$ equals **repeat** *progB* **until** $\left(\bigwedge_{o \in O} x_o = obs(o)\right)$.

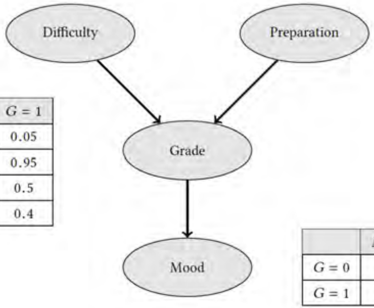
Thus: wp-reasoning of BN-programs equals exact Bayes' inference

As BN-programs are iid for every f, this is fully automatable

Exact inference by wp-reasoning

$D = 0$	$D = 1$
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

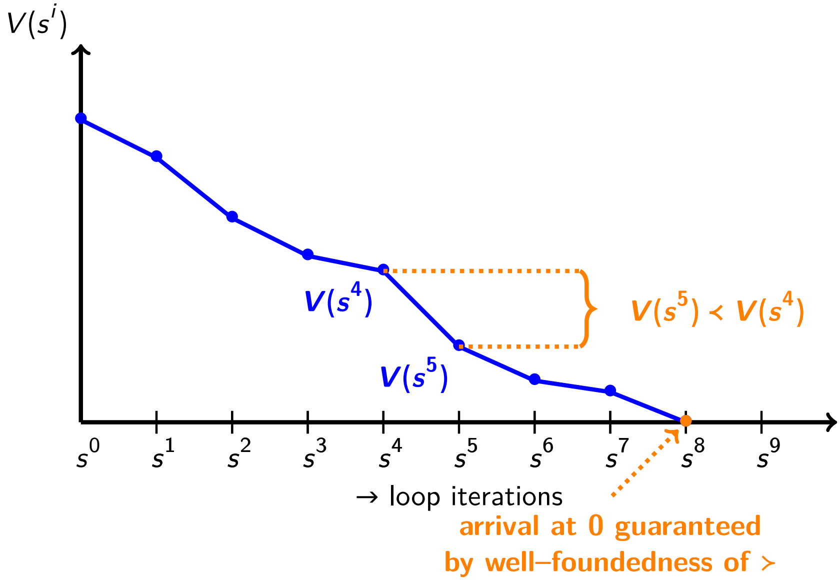
	$M = 0$	$M = 1$
$G = 0$	0.9	0.1
$G = 1$	0.3	0.7

Ergo: exact Bayesian inference can be done by wp-reasoning, e.g.,

$$wp(P_{mood}, [x_D = 0 \wedge x_G = 0 \wedge x_M = 0]) \quad = \quad \frac{Pr(D = 0, G = 0, M = 0, P = 1)}{Pr(P = 1)} \quad = \quad \mathbf{0.27}$$

Overview

Termination proofs: the classical case



Termination

[Esparza *et al.*, 2012]

"[Ordinary] termination is a purely topological property [...], but almost-sure termination is not. [...] Proving almost–sure termination requires arithmetic reasoning not offered by termination provers."

Proving a.s.-termination for a single input is Π_2-complete
(the same holds for approximate a.s.-termination)

Almost-sure termination

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

This program does not always terminate. It almost surely terminates.

Proving almost-sure termination

The symmetric random walk:

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

Is out-of-reach for many proof rules.

A loop iteration decreases x by one with probability $1/2$

This observation is enough to witness almost-sure termination!

Proving almost-sure termination

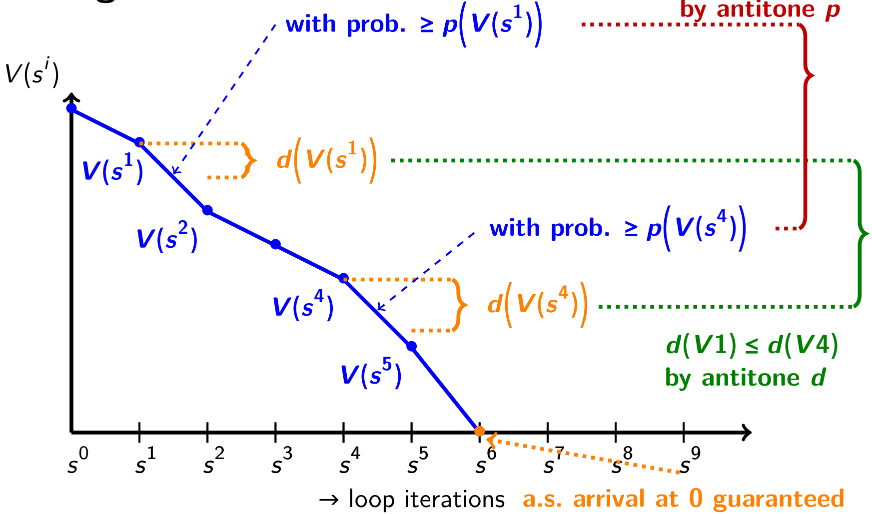
Goal: prove a.s.–termination of `while(G) P`

Ingredients:

- A supermartingale V mapping states onto non-negative reals
  - $V(s_n) \geq \mathbb{E}\{V(s_{n+1}) \mid V(s_0), \ldots, V(s_n)\}$
  - Running body `P` on state $s \models$ `G` does not increase $\mathbb{E}(V(s))$
  - Loop iteration ceases if $V(s) = 0$
- ...... and a progress condition: on each loop iteration in s^i
  - $V(s^i) = v$ decreases by $\geq d(v)$ with probability $\geq p(v)$
  - with antitone p ("probability") and d ("decrease") on V's values

Then: `while(G) P` **a.s.-terminates on every input**

Proving almost-sure termination



The closer to termination, the more V decreases and this becomes more likely

The symmetric random walk

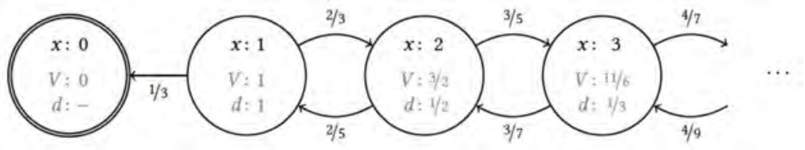
- Recall:

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

- Witnesses of almost-sure termination:
  - $V = x$
  - $p(v) = 1/2$ and $d(v) = 1$

That's all you need to prove almost-sure termination!

A symmetric-in-the-limit random walk



- Consider the program:

```
while (x > 0) { p := x/(2*x+1) ; x := x-1 [p] x := x+1 }
```

- Witnesses of almost-sure termination:
  - $V = H_x$, where H_x is x-th Harmonic number $1 + 1/2 + \ldots + 1/x$
  - $p(v) = 1/3$ and $d(v) = \begin{cases} 1/x & \text{if } v > 0 \text{ and } H_{x-1} < v \leq H_x \\ 1 & \text{if } v = 0 \end{cases}$

Expressiveness

This proof rule covers many a.s.-terminating programs

that are out-of-reach for almost all existing proof rules

Overview

Null a.s.-termination

```
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.

Positive almost-sure termination

Deciding whether a program a.s. terminates in

finitely many steps on every input, is Π^0_3-complete

Being positively a.s.-terminating is not preserved by sequential composition

Nonetheless:

Expected run-times can be determined compositionally

$ert(P, t)$ bounds P's expected run-time if P's continuation takes t time.

Expected runtime transformer

Syntax

- `skip`
- `diverge`
- `x := mu`
- `observe (G)`
- `P1 ; P2`
- `if (G) P1 else P2`
- `while(G)P`

Semantics $ert(P, t)$

- $\mathbf{1} + t$
- ∞
- $\mathbf{1} + \lambda s. \mathbb{E}_{[\![\mu]\!](s)} (\lambda v. t[x := v](s))$
- $[G] \cdot (\mathbf{1} + t)$
- $ert(P_1, ert(P_2, t))$
- $\mathbf{1} + [G] \cdot ert(P_1, t) + [\neg G] \cdot ert(P_2, t)$
- $\mu X. \mathbf{1} + ([G] \cdot ert(P, X) + [\neg G] \cdot t)$

μ is the least fixed point operator wrt. the ordering $\leq$ on run-times

and a set of proof rules [4] to get two-sided bounds on run-times of loops

[4] Certified using the Isabelle/HOL theorem prover; see [Hölzl, ITP 2016].

Run-time invariant synthesis

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

A lower ω-invariant is:

$$J_n \;=\; \mathbf{1} \;+\; \underbrace{[0 < x < n]\cdot 2x}_{\text{on iteration}} \;+\; \underbrace{[x \geq n]\cdot(2n{-}1)}_{\text{on termination}}$$

We obtain:

$$\lim_{n\to\infty}\big(\mathbf{1} + [0 < x < n]\cdot 2x + [x \geq n]\cdot(2n{-}1)\big) \;=\; \mathbf{1} + [x > 0]\cdot 2x$$

is a lower bound on the program's runtime.

Run-time invariant synthesis

```
while (c) { {c := false [0.5] c := true}; x := 2*x} ;
while (x > 0) { x := x-1 }
```

Template for a lower ω-invariant:

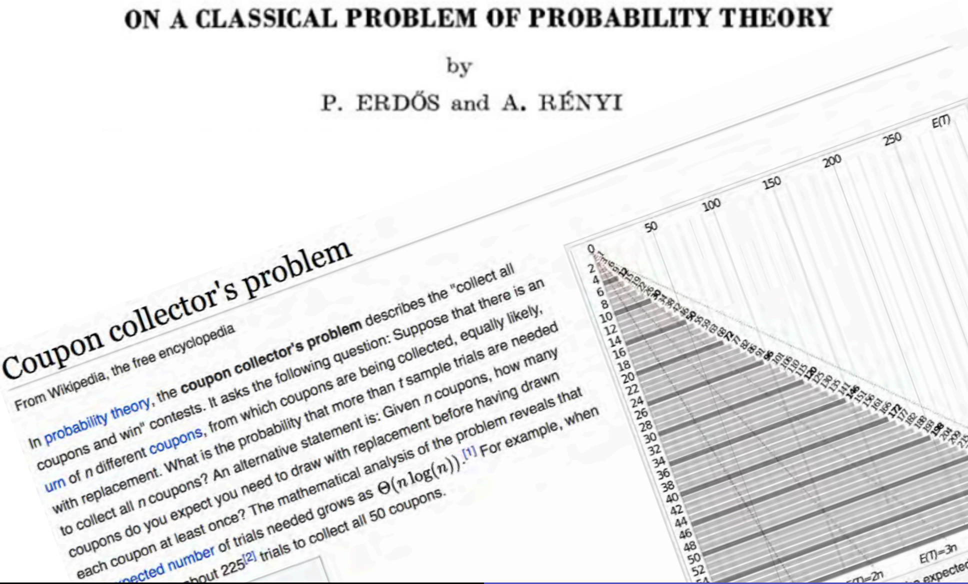
$$I_n \;=\; \mathbf{1} \;+\; \underbrace{[c \neq 1] \cdot (\mathbf{1} + [x > 0] \cdot 2x)}_{\text{on termination}} \;+\; \underbrace{[c = 1] \cdot (a_n + b_n \cdot [x > 0] \cdot 2x)}_{\text{on iteration}}$$

The derived constraints are:

$$a_0 \leq 2 \quad \text{and} \quad a_{n+1} \leq 7/2 + 1/2 \cdot a_n \quad \text{and} \quad b_0 \leq 0 \quad \text{and} \quad b_{n+1} \leq 1 + b_n$$

This admits the solution $a_n = 7 - 5/2^n$ and $b_n = n$. Then: $\lim_{n \to \infty} I_n \;=\; \infty$

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 ... out 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
}
```

Using our ert-calculus one can prove that expected run-time is $\Theta(N \cdot \log N)$.

By systematic code verification à la Floyd-Hoare. Machine checkable.

Overview

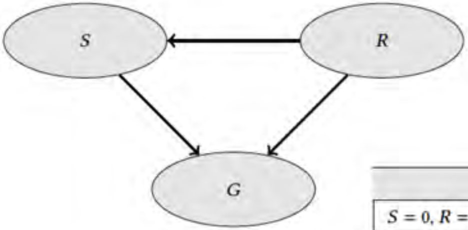
How long to sample a BN?

[Gordon, Nori, Henzinger, Rajamani, 2014]

"the main challenge in this setting [sampling-based approaches] is that many samples that are generated during execution are ultimately rejected for not satisfying the observations."

A toy Bayes' network

	$S=0$	$S=1$
$R=0$	a	$1-a$
$R=1$	0.2	0.8



$R=0$	$R=1$
a	$1-a$

	$G=0$	$G=1$
$S=0, R=0$	0.01	0.99
$S=0, R=1$	0.25	0.75
$S=1, R=0$	0.9	0.1
$S=1, R=1$	0.2	0.8

This BN is parametric (in a)

How many samples are needed on average
for a **single** iid-sample for evidence $G = 0$?

Rejection sampling

For a given Bayesian network and some evidence:

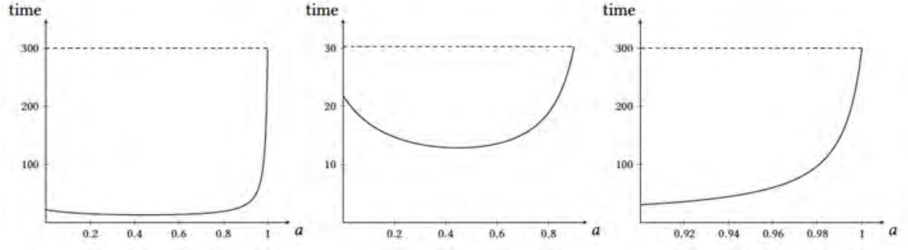
1. Sample from the joint distribution described by the BN
2. If the sample complies with the evidence, accept the sample and halt
3. If not, repeat sampling (that is: go back to step 1.)

If this procedure is applied N times, N iid-samples result.

Q: How many samples do we need on average for a **single** iid-sample?

Sampling time for example BN

Rejection sampling for $G = 0$ requires $\dfrac{200a^2 - 40a - 460}{89a^2 - 69a - 21}$ samples:



For $a \in [0.1, 0.78]$, EST is below 18; for $a \geq 0.98$, 100 samples are needed

For real-life BNs, the EST may exceed 10^{15}

Expected runtime of iid-loops

For a.s.-terminating iid-loop `while(G)P` for which every iteration runs in the same expected time, we have:

$$ert\left(\texttt{while}(\texttt{G})\texttt{P}, t\right) \;=\; \mathbf{1} + [G] \cdot \frac{\mathbf{1} + ert(P, [\neg G] \cdot t)}{1 - wp(P, [G])} + [\neg G](s) \cdot t$$

where $0/0 := 0$ and $a/0 := \infty$ for $a \neq 0$.

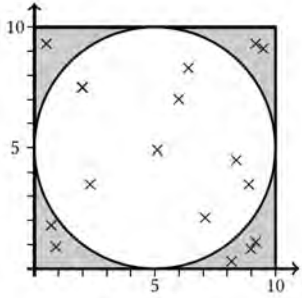
Proof: similar as for the inference (wp) using the decomposition result:

$$ert(P, t) \;=\; ert(P, \mathbf{0}) + wp(P, t)$$

No loop invariant, martingale, or metering function needed. Fully automatable.

Example: sampling within a circle

```
while ((x-5)**2 + (y-5)**2 >= 25){
    x := uniform(0..10);
    y := uniform(0..10)
}
```



This iid-loop is a.s.-terminating, and every iteration has same expected time.

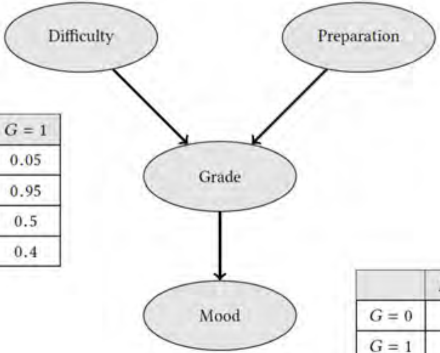
$$\text{Then:} \quad ert(P_{circle}, \mathbf{0}) \;=\; \mathbf{1} + [(x-5)^2 + (y-5)^2 \geq 25] \cdot \frac{363}{73}$$

So: $1 + {}^{363}/_{73} \approx 5.97$ operations are required on average using rejection sampling

The student's mood example

$D = 0$	$D = 1$
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

$$ert\Big(\underbrace{\texttt{repeat D; P; G; M until (P=1)}}_{\text{program of student mood's BN}}, \mathbf{0}\Big) = \frac{\mathbf{1} + ert(D; P; G; M, \mathbf{0})}{wp(D; P; G; M, [P = 1])} \approx \mathbf{23.46}$$

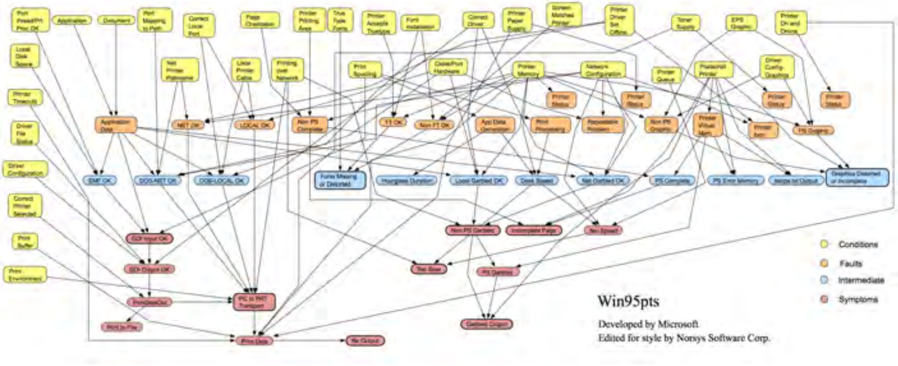
Experimental results

Benchmark BNs from `www.bnlearn.com`

BN	$\|V\|$	$\|E\|$	aMB	$\|O\|$	EST	time (s)	$\|O\|$	EST	time (s)
`hailfinder`	56	66	3.54	5	$5\,10^5$	0.63	9	$9\,10^6$	0.46
`hepar2`	70	123	4.51	1	$1.5\,10^2$	1.84	2	—	MO
`win95pts`	76	112	5.92	3	$4.3\,10^5$	0.36	12	$4\,10^7$	0.42
`pathfinder`	135	200	3.04	3	$2.9\,10^4$	31	7	∞	5.44
`andes`	223	338	5.61	3	$5.2\,10^3$	1.66	7	$9\,10^4$	0.99
`pigs`	441	592	3.92	1	$2.9\,10^3$	0.74	7	$1.5\,10^6$	1.02
`munin`	1041	1397	3.54	5	∞	1.43	10	$1.2\,10^{18}$	65

aMB = *average Markov Blanket size*, a measure of independence in BNs

Printer troubleshooting in Windows 95



Java implementation executes about 10^7 steps in a single second

For $|O|$=17, an EST of 10^{15} yields **3.6 years simulation for a single iid-sample**

Overview

Predictive probabilistic programming

Analysing probabilistic programs
at source code level, compositionally.

Some open problems:

- Completeness
- Query processing
- Invariant synthesis

Two take-home messages

Probabilistic programs are a **universal quantitative** modeling formalism:

Bayes' networks, randomised algorithms, infinite-state Markov chains, pushdown Markov chains, security mechanisms, quantum programs, programs for inexact computing . . . . . .

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

[Michael Hicks, The Programming Language Enthusiast blog, 2014]

Thanks to my co-authors!

- F. Olmedo, F. Gretz, N. Jansen, B. Kaminski, JPK, A. McIver
  Conditioning in probabilistic programming. ACM TOPLAS 2018.
- B. Kaminski, JPK.
  On the hardness of almost-sure termination. MFCS 2015.
- B. Kaminski, JPK, C. Matheja, and F. Olmedo.
  Expected run-time analysis of probabilistic programs [5]*.* ESOP 2016.
- F. Olmedo, B. Kaminski, JPK, C. Matheja.
  Reasoning about recursive probabilistic programs. LICS 2016.
- A. McIver, C. Morgan, B. Kaminski, JPK.
  A new proof rule for almost-sure termination. POPL 2018.
- K. Batz, B. Kaminski, JPK, C. Matheja.
  How long, O Bayesian network, will I sample thee? ESOP 2018.

pGCL model checking: `www.stormchecker.org`

[5] EATCS best paper award of ETAPS 2016.