CS7480: Topics in Programming Languages: Probabilistic Programming

Lecture 8: Approximate Inference

Instructor: Steven Holtzen

Place: Northeastern University

Term: Fall 2021

Course webpage:
https://www.khoury.northeastern.edu/home/sholtzen/CS7480Fall21/

Course Updates

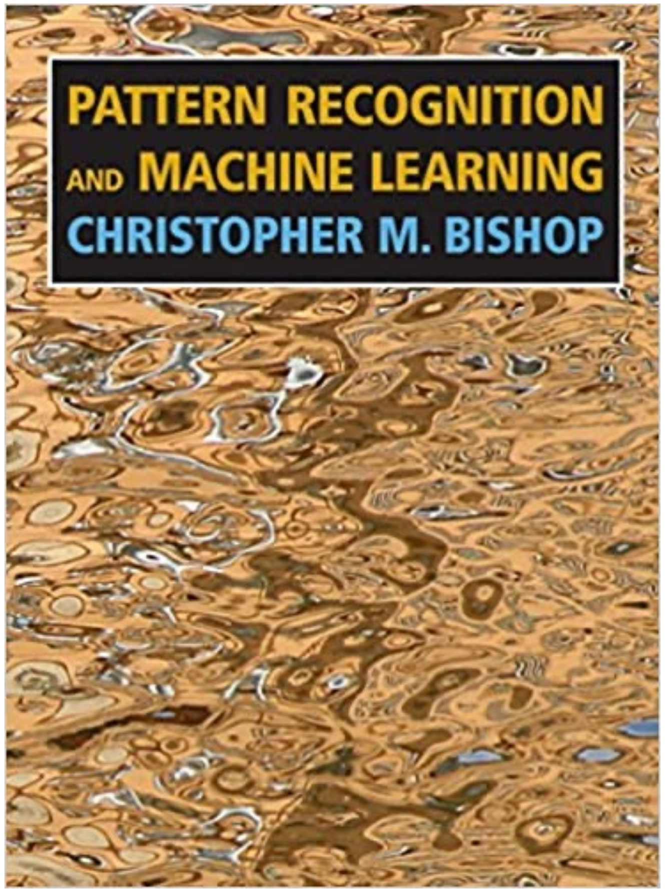
- Updated project online (dated Oct. 4 now)
  - Resolved an issue with the parser (thanks Luke)
- Reading next time: Semantics of Probabilistic Programming: A Gentle Introduction
- Pick the next paper

Approximate Inference

- *By far* the most common inference algorithm in probabilistic programming languages

- Broad families:
  - Direct methods
    - Direct sampling, importance sampling
    - Particle filtering
  - Local search methods
    - Markov-Chain Monte-Carlo
  - Optimization methods
    - Variational inference

References



- Chapter 11: Importance Sampling
- Chapter 12: Markov-Chain Monte Carlo
- Chapter 10: Variational inference

Why Approximate?

1. More expressive languages
    - Only requires the ability to draw a sample or evaluate a density on a point
    - Allows for “universal” languages that permit very complex control flow
2. ”Any-time” inference behavior
    - Trade computation time for accuracy of estimate
3. Orthogonal scaling characteristics to exact inference

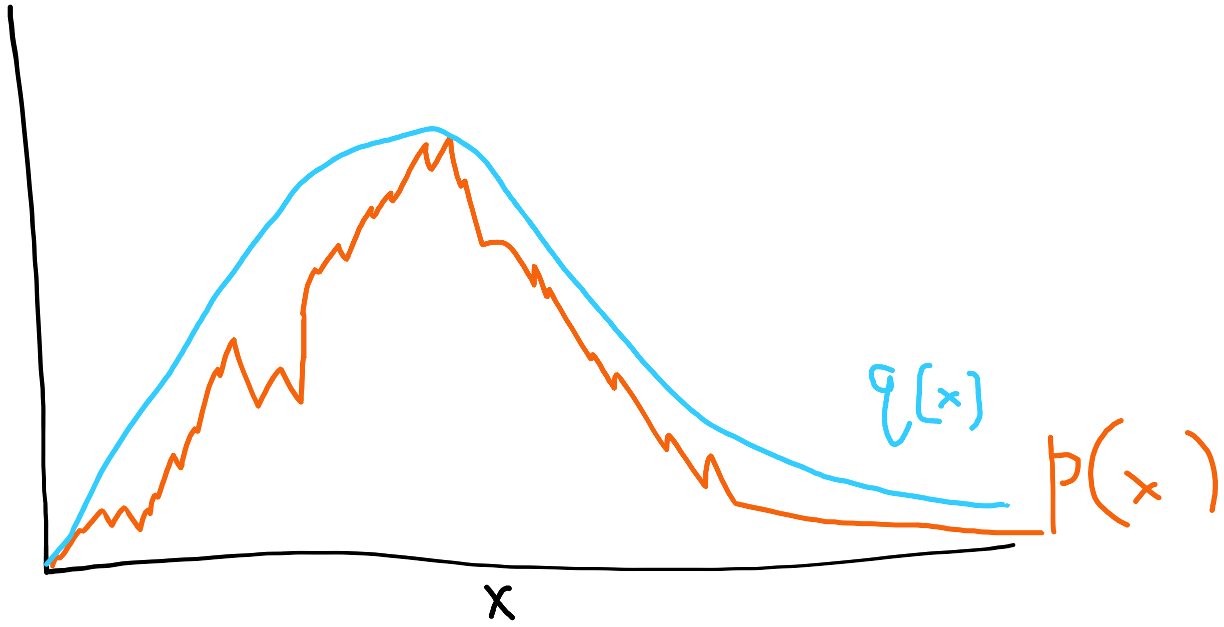
Importance Sampling

- Suppose you have a program that's hard to draw samples from directly:

```
x ~ flip 0.00001;
y ~ flip 0.000001;
observe (&& x y);
return x
```

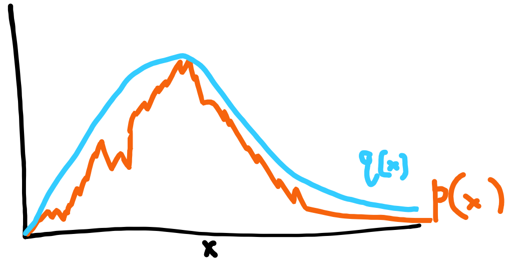
- Idea of importance sampling: find a *new program:*
  1. That is easy to sample from
  2. That is *close* (in some measure of distance) to the original program

Importance Sampling



- Assume that $p(x)$ is *hard to sample from* and $q(x)$ is *easy to sample from*
  - $q(x)$ called *proposal distribution*
  - Must contain $p(x)$ within its *support*
- *Goal:* Approximate some expectation $E_p[f]$

Importance Sampling



- Definitions
  - Let Ω be a sample space
  - $p(x)$ and $q(x)$ be distributions on Ω
  - $f : \Omega \rightarrow R$ be a random variable

$$\mathbf{E}_p[f] = \underbrace{\sum_{x \in \Omega} p(x) f(x)}_{\text{Defn of Expectation}} = \underbrace{\sum_{x \in \Omega} \underbrace{\frac{p(x)}{q(x)}}_{w(x)} q(x) f(x)}_{\text{Introduce } q} = \underbrace{\mathbf{E}_q[w(x) f(x)]}_{\text{Defn of expectation}} \approx \frac{1}{S} \sum_{x=1}^{S} w(x_s) f(x_s)$$

where $x_s \sim q(x)$

Importance Sampling Example

p

```
z ~ flip 0.7;
y ~ flip 0.4;
return z
```

q

```
z ~ flip 0.5;
y ~ flip 0.5;
return z
```

z	y	$p(z,y)$	$q(z,y)$	$w(z,y)$
1	1	0.7×0.4	0.5×0.5	$\frac{0.7\times 0.4}{0.5\times 0.5} = 1.12$
0	0	0.3×0.6	0.5×0.5	$\frac{0.3\times 0.6}{0.5\times 0.5} = 0.72$
1	0	0.7×0.6	0.5×0.5	$\frac{0.7\times 0.6}{0.5\times 0.5} = 1.68$

Then, query prob is

$$\approx \frac{1}{3}(1.12 + 1.68) \approx 0.93$$

To see that it's correct asymptotically, note that $1.12 \times 0.25 + 1.68 \times 0.25 = 0.7$

Self-Normalized Importance Sampling

- It is often the case that we only know how to compute the *unnormalized probability* of $p(x)$

```
x ~ flip 0.00001;
y ~ flip 0.000001;
observe (&& x y);
return x
```

$$\hat{P}r(x = T, y = T) = 0.00001 \times 0.000001$$

- We can still use importance sampling, but we must estimate the normalizing constant and the query simultaneously

Self-Normalized Importance Sampling

- Let $\hat{p}(x) = \frac{1}{Z_p} p(x)$

- Then, for $x_s \sim q(x)$,

$$E[f] \approx \underbrace{\frac{1}{\sum_s \hat{w}(x_s)}}_{\text{Estimate of } Z_p} \underbrace{\sum_{s=1}^{S} \hat{w}(x_s) f(x_s)}_{\text{Regular IS estimator}}, \qquad \text{where } \hat{w}(x_s) = \frac{\hat{p}(x_s)}{q(x_s)}.$$

SNIS Example: Likelihood Weighted

p

```
x ~ flip 0.00001;
y ~ flip 0.000001;
observe (|| x y);
return x
```

q

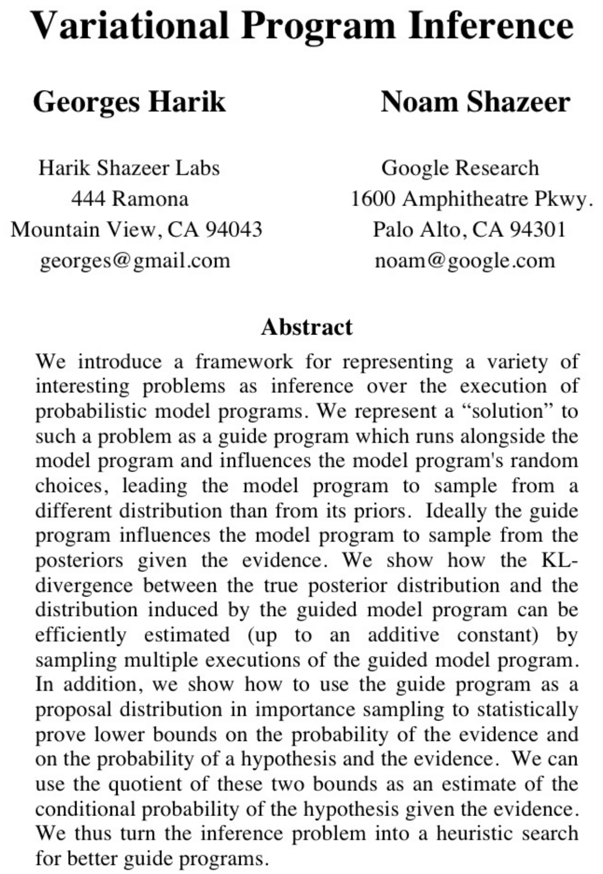
```
x ~ flip 0.00001;
y ~ flip 0.000001;
return x
```

z	y	$\hat{p}(x,y)$	$q(x,y)$	$\hat{w}(x,y)$
1	1	0.000000001	0.000000001	$\frac{0.000000001}{0.00000001} = 1$
0	0	0	0.999989	0
0	1	0.00001	0.00001	1

Essentially direct sampling where you simultaneously estimate Z and the query

$$E[f] \approx \frac{1}{\sum_s \hat{w}(x_s)} \sum_{s=1}^{S} \hat{w}(x_s) f(x_s), \qquad \text{where } \hat{w}(x_s) = \frac{\hat{p}(x_s)}{q(x_s)}.$$

Guide Programs



Variational Program Inference

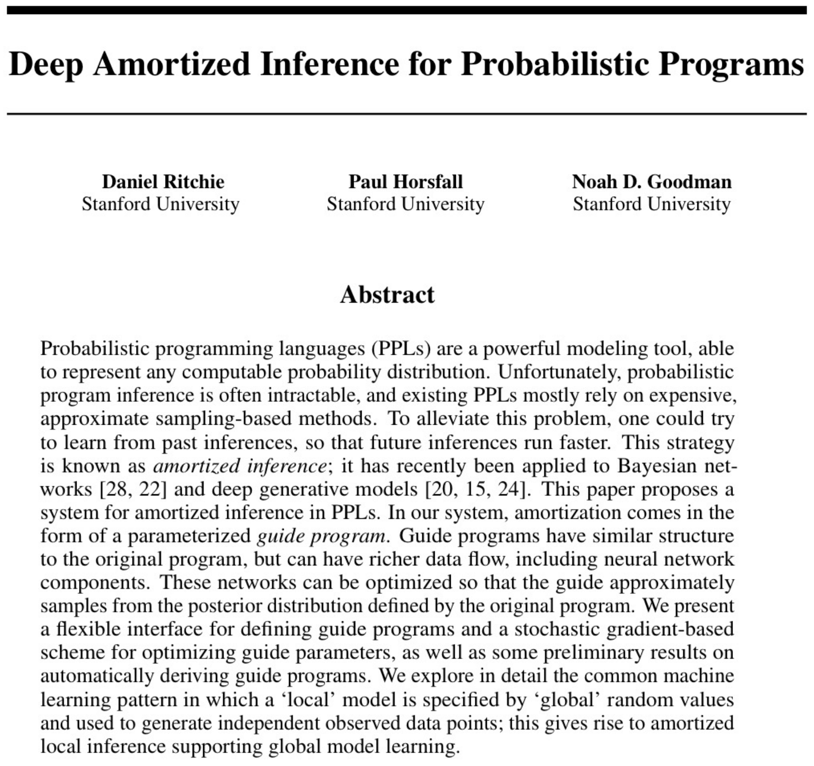
Georges Harik
Harik Shazeer Labs
444 Ramona
Mountain View, CA 94043
georges@gmail.com

Noam Shazeer
Google Research
1600 Amphitheatre Pkwy.
Palo Alto, CA 94301
noam@google.com

Abstract

We introduce a framework for representing a variety of interesting problems as inference over the execution of probabilistic model programs. We represent a "solution" to such a problem as a guide program which runs alongside the model program and influences the model program's random choices, leading the model program to sample from a different distribution than from its priors. Ideally the guide program influences the model program to sample from the posteriors given the evidence. We show how the KL-divergence between the true posterior distribution and the distribution induced by the guided model program can be efficiently estimated (up to an additive constant) by sampling multiple executions of the guided model program. In addition, we show how to use the guide program as a proposal distribution in importance sampling to statistically prove lower bounds on the probability of the evidence and on the probability of a hypothesis and the evidence. We can use the quotient of these two bounds as an estimate of the conditional probability of the hypothesis given the evidence. We thus turn the inference problem into a heuristic search for better guide programs.

Harik, Georges, and Noam Shazeer. "Variational program inference." arXiv preprint arXiv:1006.0991 (2010).



Deep Amortized Inference for Probabilistic Programs

Daniel Ritchie
Stanford University

Paul Horsfall
Stanford University

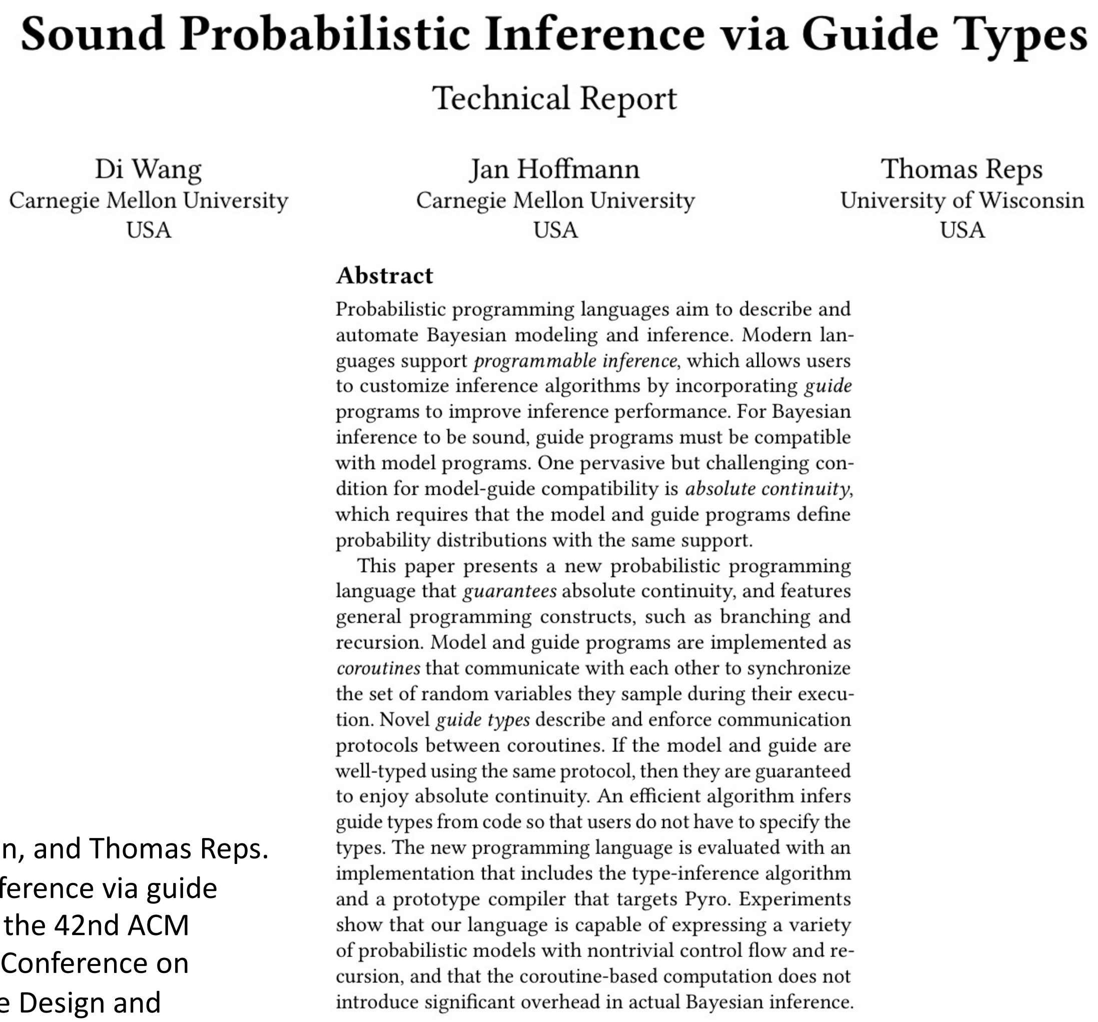
Noah D. Goodman
Stanford University

Abstract

Probabilistic programming languages (PPLs) are a powerful modeling tool, able to represent any computable probability distribution. Unfortunately, probabilistic program inference is often intractable, and existing PPLs mostly rely on expensive, approximate sampling-based methods. To alleviate this problem, one could try to learn from past inferences, so that future inferences run faster. This strategy is known as *amortized inference*; it has recently been applied to Bayesian networks [28, 22] and deep generative models [20, 15, 24]. This paper proposes a system for amortized inference in PPLs. In our system, amortization comes in the form of a parameterized *guide program*. Guide programs have similar structure to the original program, but can have richer data flow, including neural network components. These networks can be optimized so that the guide approximately samples from the posterior distribution defined by the original program. We present a flexible interface for defining guide programs and a stochastic gradient-based scheme for optimizing guide parameters, as well as some preliminary results on automatically deriving guide programs. We explore in detail the common machine learning pattern in which a 'local' model is specified by 'global' random values and used to generate independent observed data points; this gives rise to amortized local inference supporting global model learning.

Ritchie, Daniel, Paul Horsfall, and Noah D. Goodman. "Deep amortized inference for probabilistic programs." arXiv preprint arXiv:1610.05735 (2016).

Guide Programs



Sound Probabilistic Inference via Guide Types

Technical Report

Di Wang
Carnegie Mellon University
USA

Jan Hoffmann
Carnegie Mellon University
USA

Thomas Reps
University of Wisconsin
USA

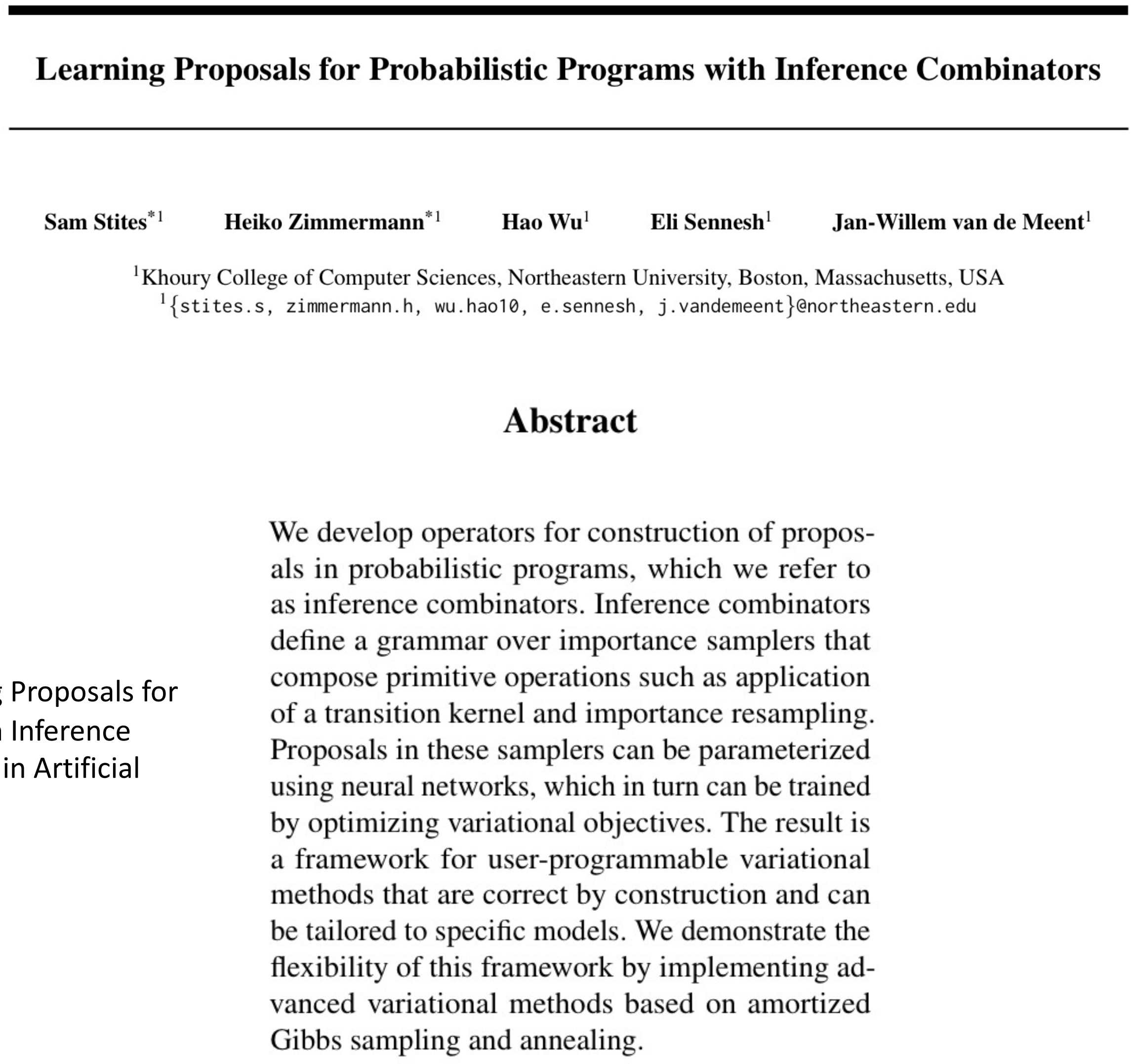
Abstract

Probabilistic programming languages aim to describe and automate Bayesian modeling and inference. Modern languages support *programmable inference*, which allows users to customize inference algorithms by incorporating *guide* programs to improve inference performance. For Bayesian inference to be sound, guide programs must be compatible with model programs. One pervasive but challenging condition for model-guide compatibility is *absolute continuity*, which requires that the model and guide programs define probability distributions with the same support.

This paper presents a new probabilistic programming language that *guarantees* absolute continuity, and features general programming constructs, such as branching and recursion. Model and guide programs are implemented as *coroutines* that communicate with each other to synchronize the set of random variables they sample during their execution. Novel *guide types* describe and enforce communication protocols between coroutines. If the model and guide are well-typed using the same protocol, then they are guaranteed to enjoy absolute continuity. An efficient algorithm infers guide types from code so that users do not have to specify the types. The new programming language is evaluated with an implementation that includes the type-inference algorithm and a prototype compiler that targets Pyro. Experiments show that our language is capable of expressing a variety of probabilistic models with nontrivial control flow and recursion, and that the coroutine-based computation does not introduce significant overhead in actual Bayesian inference.

Wang, Di, Jan Hoffmann, and Thomas Reps. "Sound probabilistic inference via guide types." Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. 2021.

Guide Programs



Learning Proposals for Probabilistic Programs with Inference Combinators

Sam Stites[*1] **Heiko Zimmermann**[*1] **Hao Wu**[1] **Eli Sennesh**[1] **Jan-Willem van de Meent**[1]

[1]Khoury College of Computer Sciences, Northeastern University, Boston, Massachusetts, USA
[1]{stites.s, zimmermann.h, wu.hao10, e.sennesh, j.vandemeent}@northeastern.edu

Abstract

We develop operators for construction of proposals in probabilistic programs, which we refer to as inference combinators. Inference combinators define a grammar over importance samplers that compose primitive operations such as application of a transition kernel and importance resampling. Proposals in these samplers can be parameterized using neural networks, which in turn can be trained by optimizing variational objectives. The result is a framework for user-programmable variational methods that are correct by construction and can be tailored to specific models. We demonstrate the flexibility of this framework by implementing advanced variational methods based on amortized Gibbs sampling and annealing.

Stites, Sam, et al. "Learning Proposals for Probabilistic Programs with Inference Combinators." Uncertainty in Artificial Intelligence, 2021.

Importance Sampling Summary

- When does it work well?
  - If you have a good proposal (close to the posterior around the function over which you are computing the expectation)
- When does it fail?
  - If your proposal is “far away” from your true distribution

Local-Search Methods

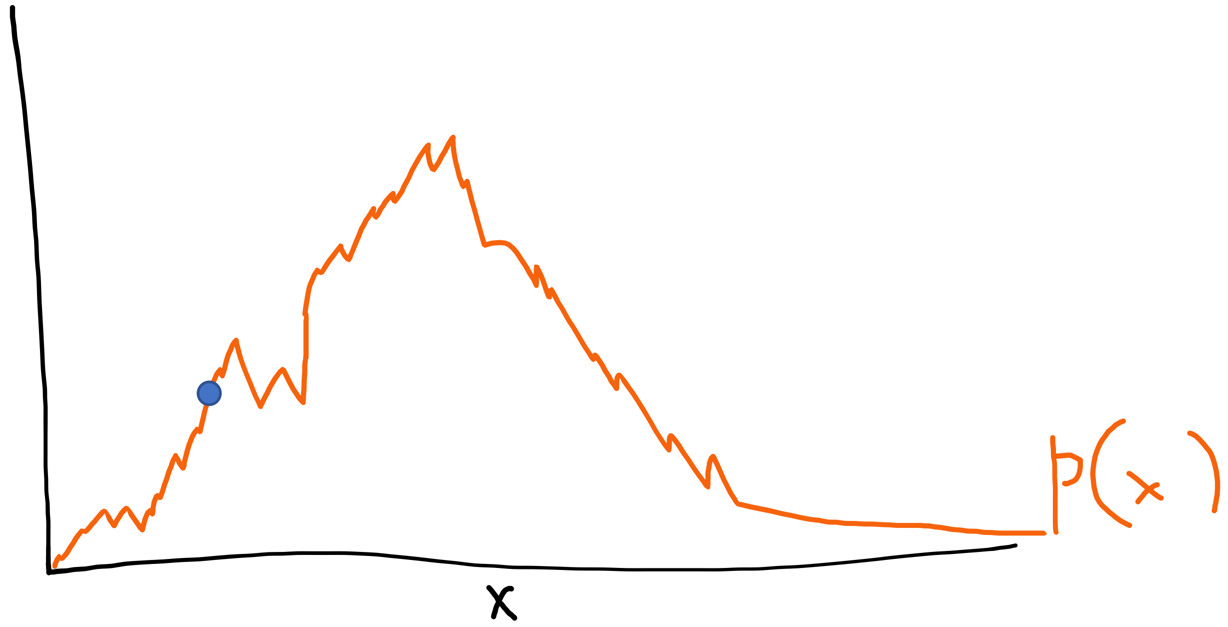
Markov-Chain Monte Carlo

Direct vs. Local

- The approximate inference methods we've been looking at so far are "direct" methods: each sample is *independent* from the previous

- Local methods relax this assumption by allowing samples to be *dependent* on one another

Markov-Chain Monte Carlo



- Idea: sample the state space by *picking a point* and *making local moves*
  - If we make moves in the right way, we can ensure that eventually this point will be distributed the same as $p(x)$

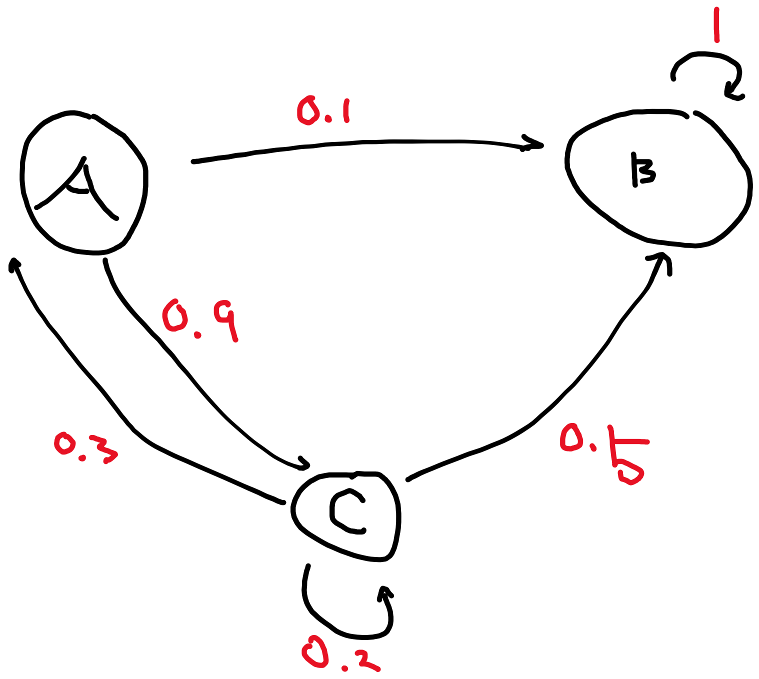
Local Search Intuition

- Why might local search be a good idea? *If we know where to start!*

```
x ~ flip 0.000000000001;
y ~ flip 0.000000000000001;
observe (|| x y);
return x
```

- Start at the state {x=T, y=T} and make *local moves* around this state
  - For instance, changing one variable at a time

Markov Chains



- Initial distribution on states
- Let $T(\,x' \mid x\,)$ denote probability of transitioning from state x to x'
- Describes a distribution over states ($\Omega = \{A, B, C\}$) at time t, denoted $\Pr_t(x)$
- A *steady state* is defined:

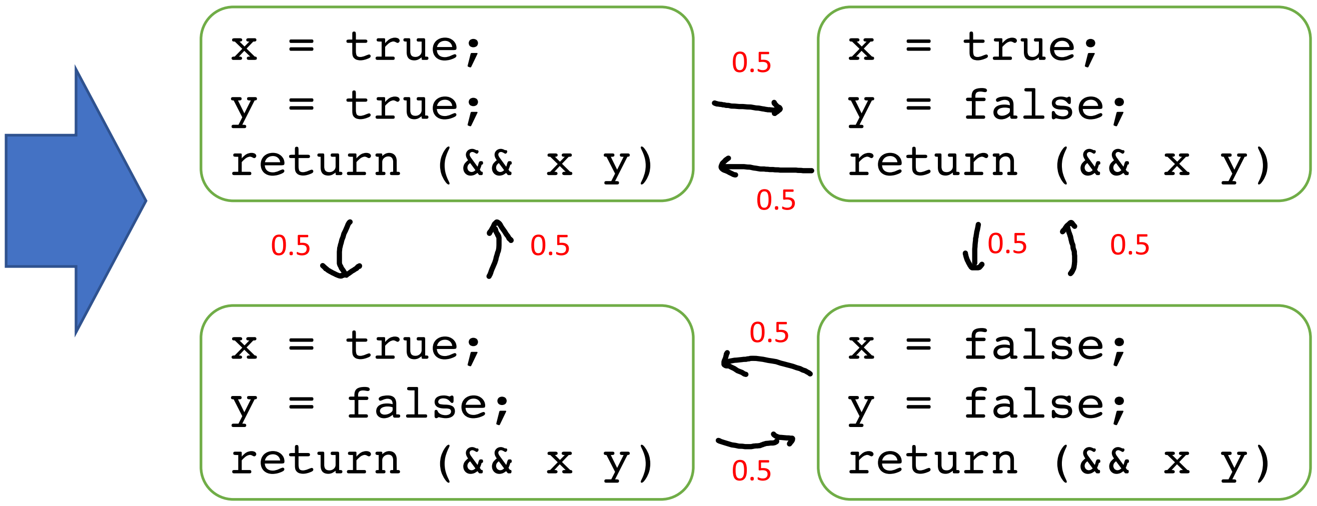
$$p^*(x') = \sum_x T(\,x' \mid x\,)p^*(x)$$

- Steady state does not always exist, is not always unique
  - In this case it's $\Pr(X = B) = 1$

Markov-Chain Monte Carlo

- Idea: Design a Markov chain whose stationary distribution is equal to the distribution from which you want to sample
  - To approximately draw a sample, run the chain for some finite amount of time

```
x ~ flip 0.5;
y ~ flip 0.5;
return (&& x y)
```



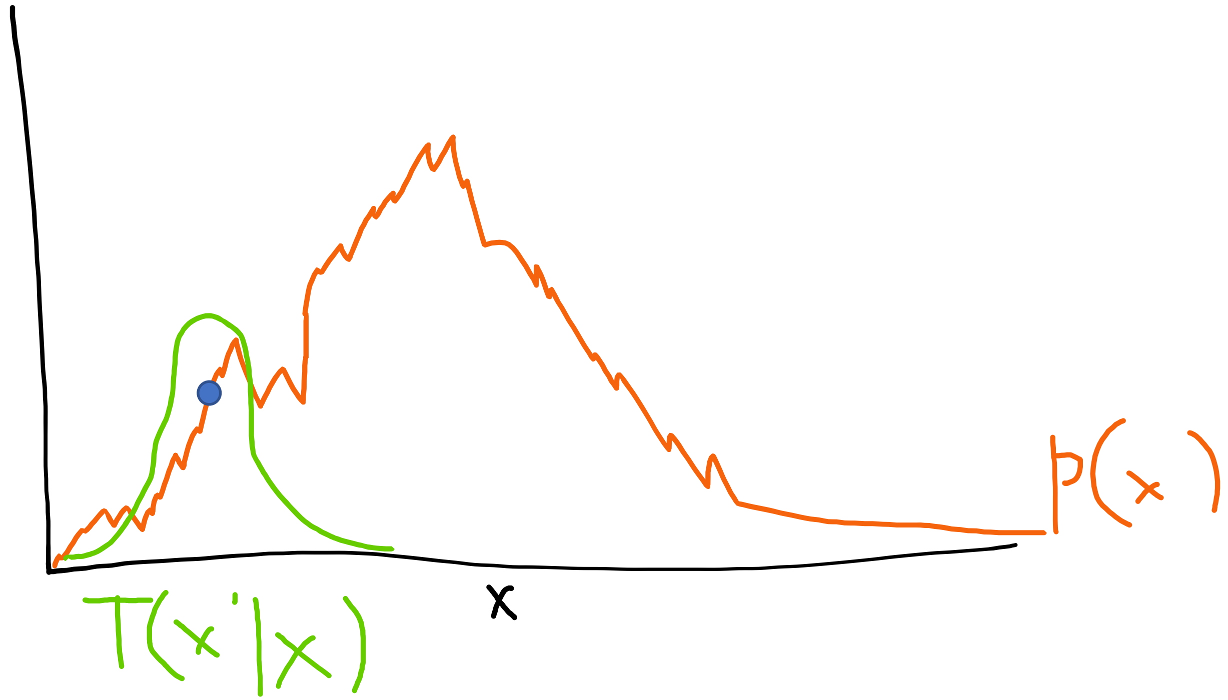
```
x = true;
y = true;
return (&& x y)
```

```
x = true;
y = false;
return (&& x y)
```

```
x = true;
y = false;
return (&& x y)
```

```
x = false;
y = false;
return (&& x y)
```

Markov-Chain Monte Carlo



Detailed Balance

- Suppose we want $p(x)$ to be the stationary distribution
- We can choose $T(x \mid x')$ so that the following *detailed balance* condition holds:

$$p(x)T(\, x \mid x' \,) = p(x')T(\, x' \mid x \,)$$

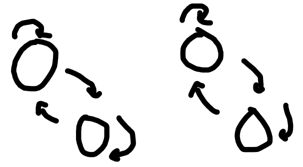
- We can prove that any transition matrix that satisfies this has $p(x)$ as its stationary dist:

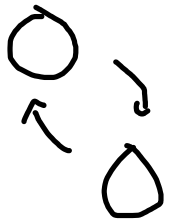
$$\sum_x p(x)T(x' \mid x) = \sum_x p(x')T(x \mid x')$$

$$= p(x') \sum_x T(\, x \mid x' \,) = p(x').$$

Uniqueness of Fixed Points

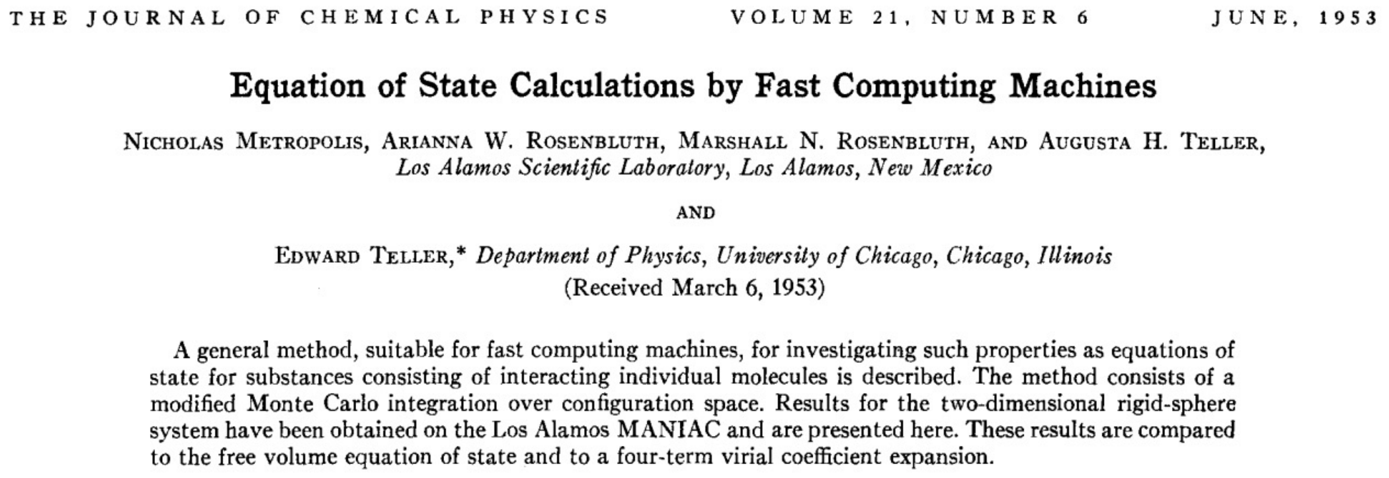
- Detailed balance is enough to ensure that $p(x)$ is *a* stationary dist, but not strong enough to ensure it's the *only* one

- For uniqueness and existence, chain must be:
  - **Irreducible**: No disconnected states
  
  - **Aperiodic:** no alternation between modes



Metropolis Algorithm

- How do we build a transition T that satisfies detailed balance, aperiodicity, and irreducibility?



THE JOURNAL OF CHEMICAL PHYSICS VOLUME 21, NUMBER 6 JUNE, 1953

Equation of State Calculations by Fast Computing Machines

NICHOLAS METROPOLIS, ARIANNA W. ROSENBLUTH, MARSHALL N. ROSENBLUTH, AND AUGUSTA H. TELLER,
Los Alamos Scientific Laboratory, Los Alamos, New Mexico

AND

EDWARD TELLER,* *Department of Physics, University of Chicago, Chicago, Illinois*
(Received March 6, 1953)

A general method, suitable for fast computing machines, for investigating such properties as equations of state for substances consisting of interacting individual molecules is described. The method consists of a modified Monte Carlo integration over configuration space. Results for the two-dimensional rigid-sphere system have been obtained on the Los Alamos MANIAC and are presented here. These results are compared to the free volume equation of state and to a four-term virial coefficient expansion.

MANIAC I
Los Alamos, 1953
© Corbis Images

Metropolis Algorithm

- Provide a transition $q(x' \mid x)$
  - Symmetric: $q(x' \mid x) = q(x \mid x')$
  - Full support: For any $x, x' \in \Omega$, $q(x' \mid x) > 0$

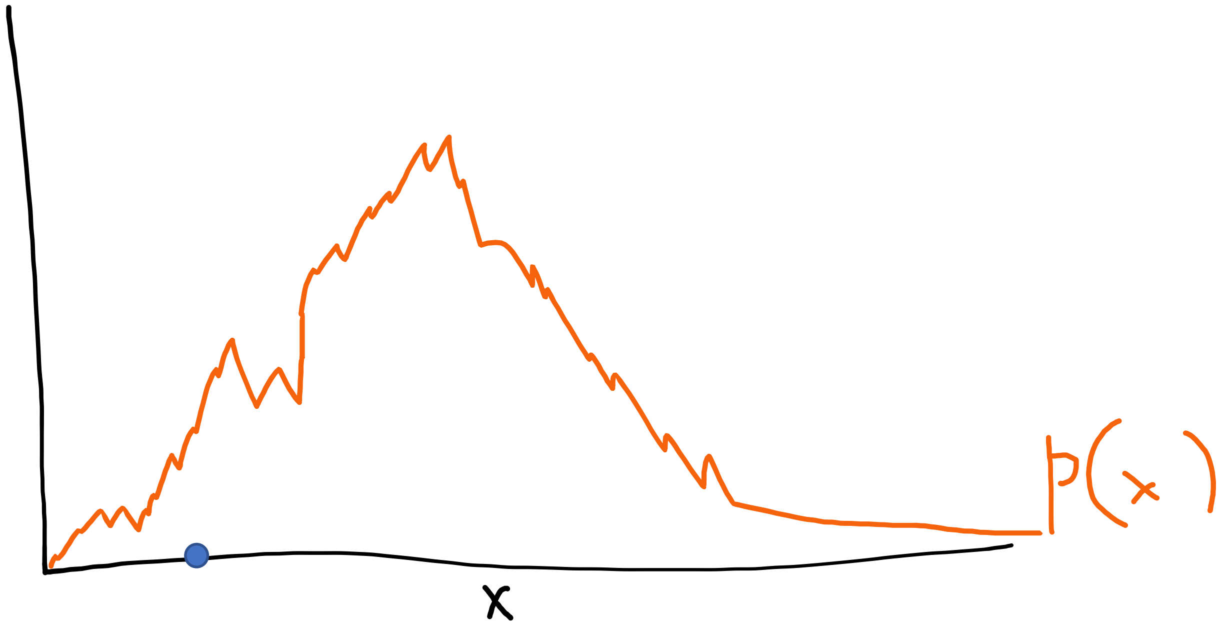
Very common choice of $q(\cdot \mid x)$ Gaussian w/ mean at x

- "Fix it up" so that it's *guaranteed* to be a valid Markov chain for sampling from $p(x)$
  - Starting from an initial state $x \in \Omega$
  1. Propose a sample $x' \sim q(\cdot \mid x)$
  2. Accept x' with probability $\min\left(1, \frac{\hat{p}(x')}{\hat{p}(x)}\right)$

- This chain is guaranteed to be aperiodic, irreducible, and have stationary distribution $p(x)$

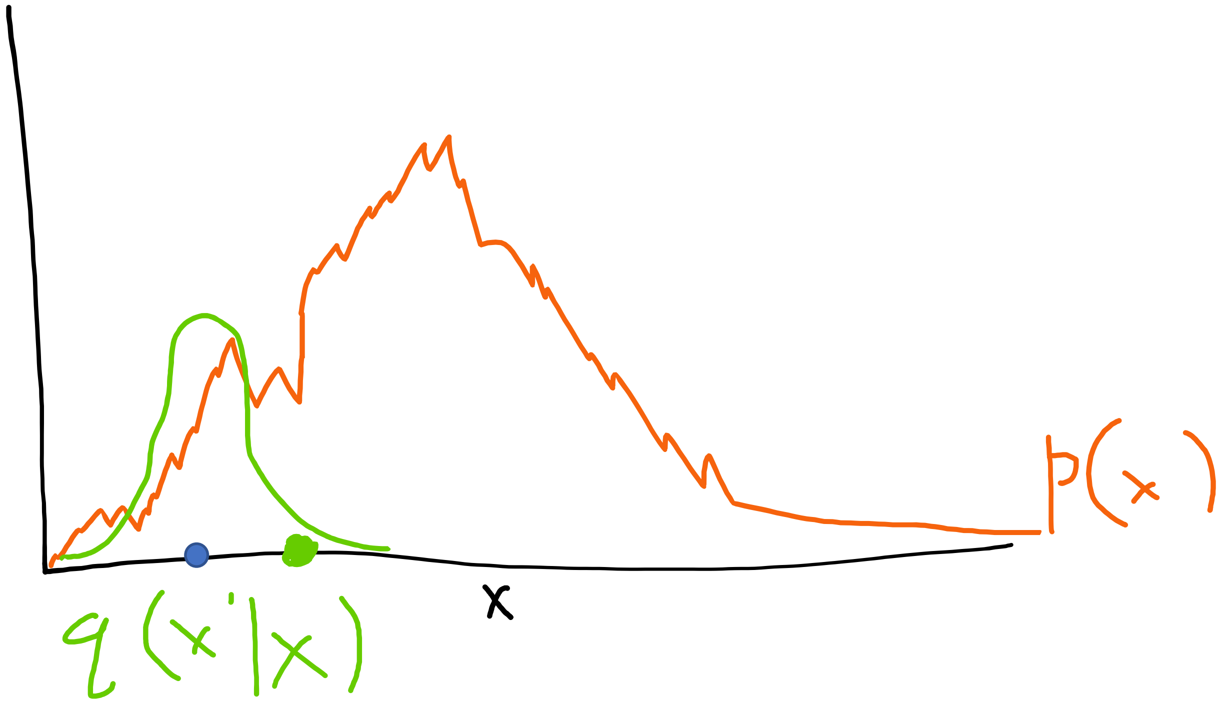
Metropolis Algorithm

- Initial Point



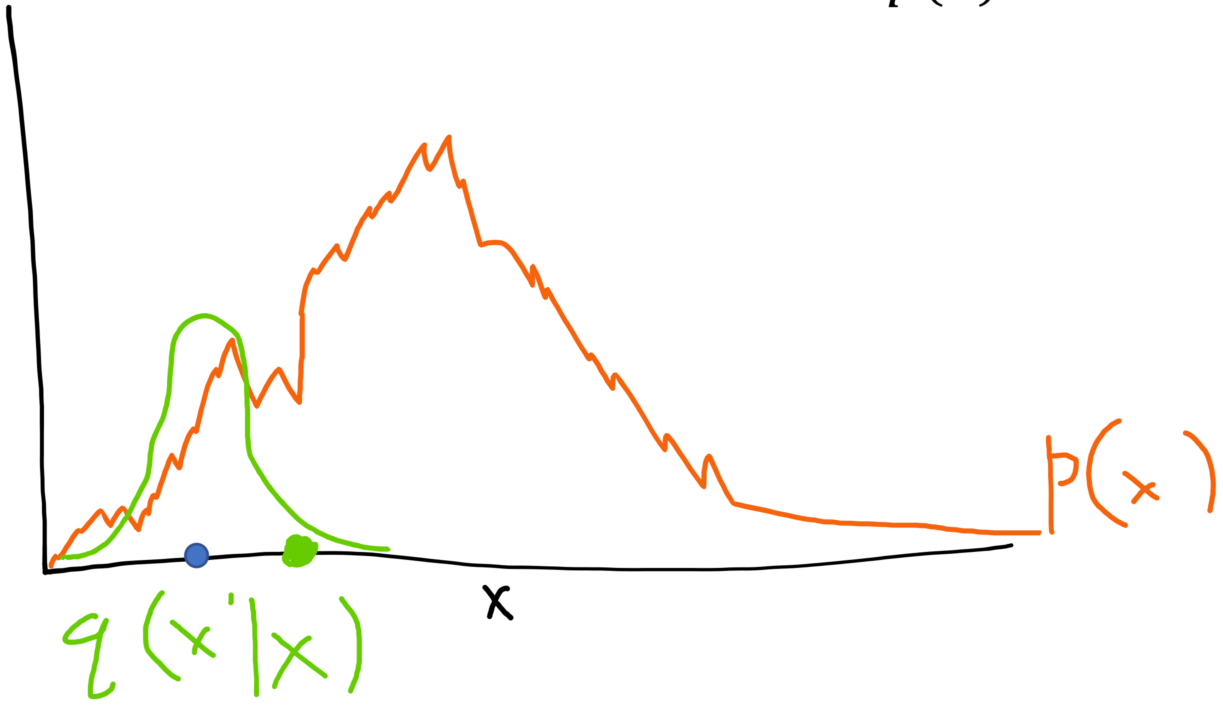
Metropolis Algorithm

- Draw a sample from proposal q



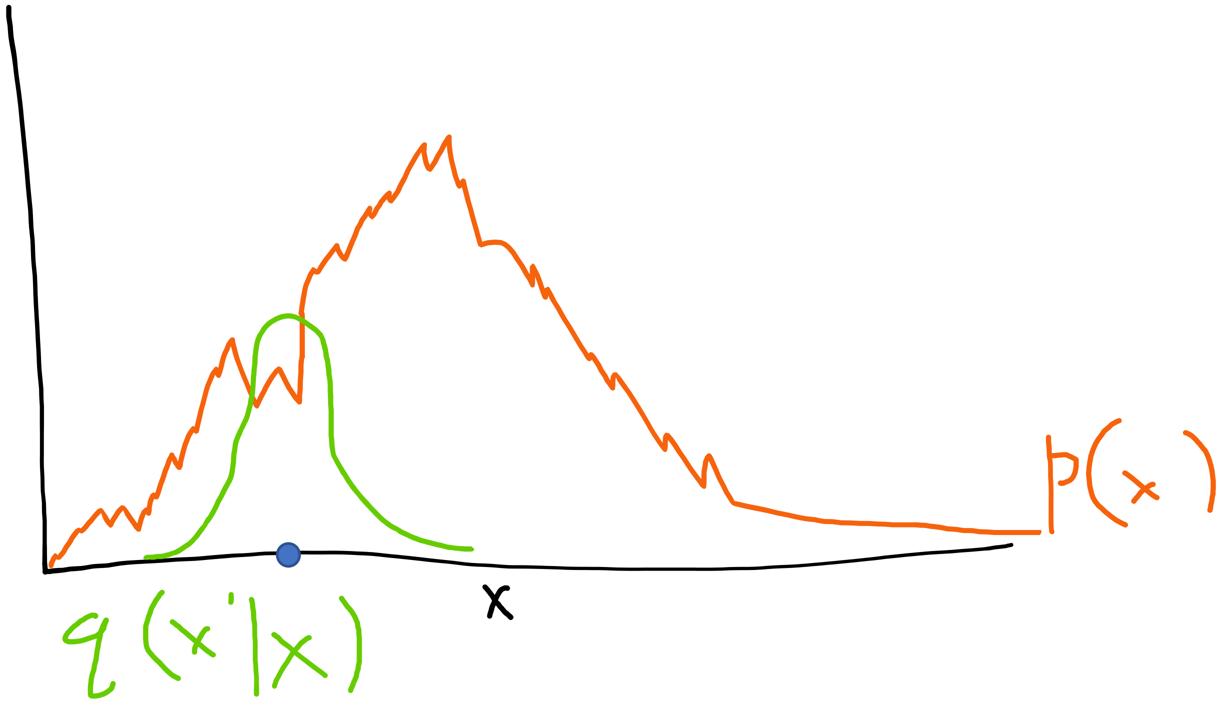
Metropolis Algorithm

- Accept x' with probability $\min\left(1, \frac{\hat{p}(x')}{\hat{p}(x)}\right)$



Metropolis Algorithm

- Propose new point

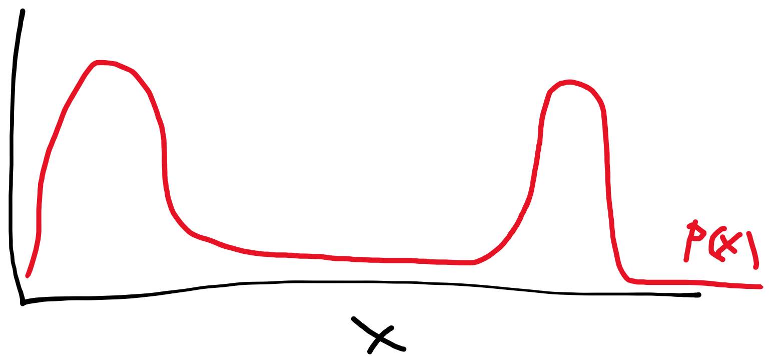


Metropolis Detailed Balance

$$\begin{aligned}
p(x)T(x' \mid x) &= p(x)q(x' \mid x)\min\left(1, \frac{\hat{p}(x')}{\hat{p}(x)}\right) \\
&= \min\left(p(x)q(x' \mid x),\ \ p(x')q(x' \mid x)\right) && \text{Because } \hat{p} = \frac{1}{Z}p \\
&= \min\left(p(x)q(x \mid x'),\ \ p(x')q(x \mid x')\right) && \text{Symmetry} \\
&= \min\left(p(x')q(x \mid x'),\ \ p(x)q(x \mid x')\right) && \text{Reordering} \\
&= p(x')q(x \mid x')\min\left(1, \frac{\hat{p}(x)}{\hat{p}(x')}\right) \\
&= p(x)T(x \mid x')
\end{aligned}$$

Challenges for MCMC

- Slow mixing (aka. "random walk behavior")
    - Multi-modal distributions
    - Distributions with "flat regions"

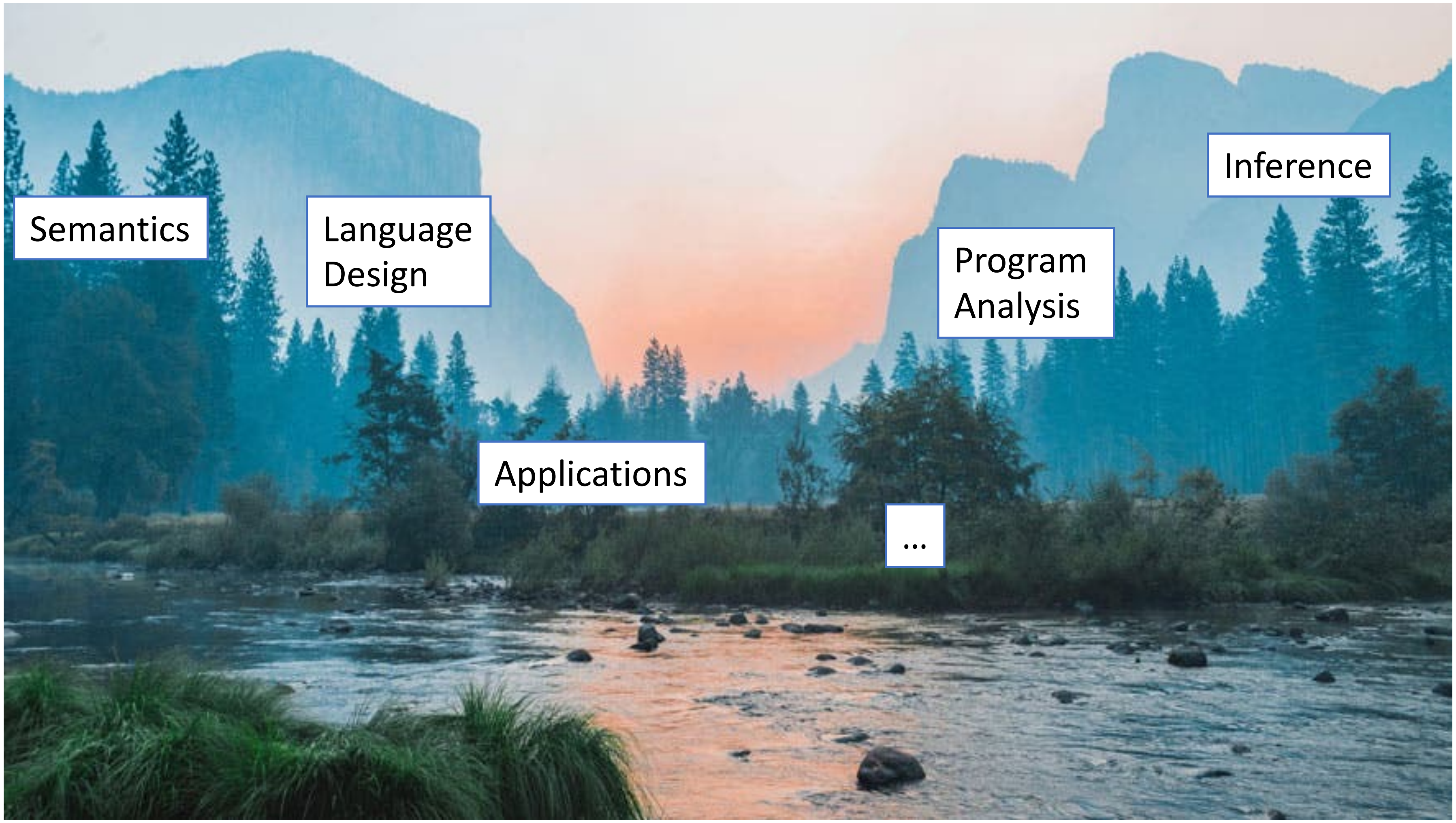


- Choice of proposal q is ***extremely important***

Ways of Choosing q

- Gibbs sampling (coordinate-wise updates)
  - For a joint distribution $p(\{x_i\})$, define q as
    - (1) choose a random variable x_i
    - (2) re-sample from $p(x_i \mid x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n)$
- Momentum-based methods (Hamiltonian Monte-Carlo, NUTS)
  - Idea: make ”bigger steps” in boring regions (high acceptance probability), slow down in trickier regions (low acceptance probability)

Conclusion


Semantics
Language Design
Applications
...
Program Analysis
Inference