Math ∩ Programming

Markov Chain Monte Carlo Without all the Bullshit

Posted on April 6, 2015 by j2kun

I have a little secret: I don't like the terminology, notation, and style of writing in statistics. I find it unnecessarily complicated. This shows up when trying to read about Markov Chain Monte Carlo methods. Take, for example, the abstract to the Markov Chain Monte Carlo article in the Encyclopedia of Biostatistics. (http://onlinelibrary.wiley.com/doi/10.1002/0470011815.b2a14021/abstract?deniedAccessCustomisedMessage=&userIsAuthenticated=false)

> *Markov chain Monte Carlo (MCMC) is a technique for estimating by simulation the expectation of a statistic in a complex model. Successive random selections form a Markov chain, the stationary distribution of which is the target distribution. It is particularly useful for the evaluation of posterior distributions in complex Bayesian models. In the Metropolis–Hastings algorithm, items are selected from an arbitrary "proposal" distribution and are retained or not according to an acceptance rule. The Gibbs sampler is a special case in which the proposal distributions are conditional distributions of single components of a vector parameter. Various special cases and applications are considered.*

I can only vaguely understand what the author is saying here (and really only because I know ahead of time what MCMC is). There are certainly references to more advanced things than what I'm going to cover in this post. But it seems very difficult to find an explanation of Markov Chain Monte Carlo *without* superfluous jargon. The "bullshit" here is the implicit claim of an author that such jargon is needed. Maybe it is to explain advanced applications (like attempts to do "inference in Bayesian networks"), but it is certainly not needed to define or analyze the basic ideas.

So to counter, here's my own explanation of Markov Chain Monte Carlo, inspired by the treatment of John Hopcroft and Ravi Kannan (http://research.microsoft.com/en-US/people/kannan/book-no-solutions-aug-21-2014.pdf).

The Problem is Drawing from a Distribution

Markov Chain Monte Carlo is a technique to solve the problem of *sampling from a complicated distribution.* Let me explain by the following imaginary scenario. Say I have a magic box which can estimate probabilities of baby names very well. I can give it a string like "Malcolm" and it will tell me the exact probability $p_{\textup{Malcolm}}$ that you will choose this name for your next child. So there's a distribution D over all names, it's very specific to your preferences, and for the sake of argument say this distribution is fixed and you don't get to tamper with it.

Now comes the problem: I want to *efficiently draw* a name from this distribution D. This is the problem that Markov Chain Monte Carlo aims to solve. Why is it a problem? Because I have no idea what process you use to pick a name, so I can't simulate that process myself. Here's another method you could try: generate a name x uniformly at random, ask the machine for p_x, and then flip a biased coin with probability p_x and use x if the coin lands heads. The problem with this is that there are exponentially many names! The variable here is the number of bits needed to write down a name $n = |x|$. So either the probabilities p_x will be exponentially small and I'll be flipping for a very long time to get a single name, or else there will only be a few names with nonzero probability and it will take me exponentially many draws to find them. Inefficiency is the death of me.

So this is a serious problem! Let's restate it formally just to be clear.

Definition (The sampling problem): Let D be a distribution over a finite set X. You are given black-box access to the probability distribution function $p(x)$ which outputs the probability of drawing $x \in X$ according to D. Design an efficient randomized algorithm A which outputs an element of X so that the probability of outputting x is approximately $p(x)$. More generally, output a sample of elements from X drawn according to $p(x)$.

Assume that A has access to only fair random coins, though this allows one to efficiently simulate flipping a biased coin of any desired probability (https://jeremykun.com/2014/02/12/simulating-a-biased-coin-with-a-fair-coin/).

Notice that with such an algorithm we'd be able to do things like estimate the expected value of some random variable $f : X \to \mathbb{R}$. We could take a large sample $S \subset X$ via the solution to the sampling problem, and then compute the average value of f on that sample. This is what a Monte Carlo method does when sampling is easy. In fact, the Markov Chain solution to the sampling problem will allow us to do the sampling *and* the estimation of $\mathbb{E}(f)$ in one fell swoop if you want.

But the core problem is really a sampling problem, and "Markov Chain Monte Carlo" would be more accurately called the "Markov Chain Sampling Method." So let's see why a Markov Chain could possibly help us.

Random Walks, the "Markov Chain" part of MCMC

Markov Chain is essentially a fancy term for a random walk on a graph.

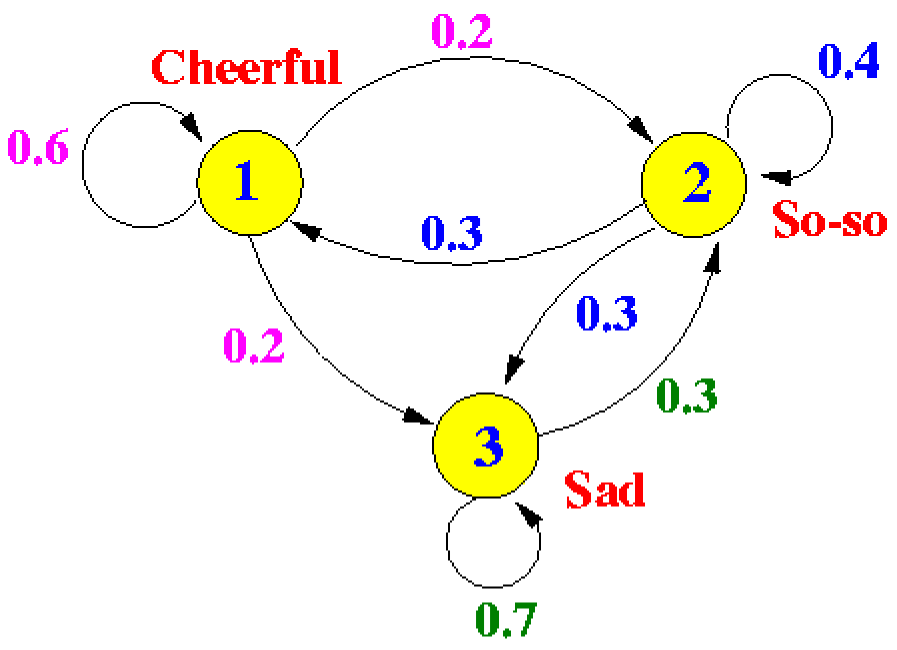
You give me a directed graph $G = (V, E)$, and for each edge $e = (u, v) \in E$ you give me a number $p_{u,v} \in [0, 1]$. In order to make a random walk make sense, the $p_{u,v}$ need to satisfy the following constraint:

For any vertex $x \in V$, the set all values $p_{x,y}$ on outgoing edges (x, y) must sum to 1, i.e. form a probability distribution.

If this is satisfied then we can take a random walk on G according to the probabilities as follows: start at some vertex x_0. Then pick an outgoing edge at random according to the probabilities on the outgoing edges, and follow it to x_1. Repeat if possible.

I say "if possible" because an arbitrary graph will not necessarily have any outgoing edges from a given vertex. We'll need to impose some additional conditions on the graph in order to apply random walks to Markov Chain Monte Carlo, but in any case the idea of randomly walking is well-defined, and we call the whole object $(V, E, \{p_e\}_{e \in E})$ a *Markov chain.*

Here is an example where the vertices in the graph correspond to emotional states.



(https://jeremykun.files.wordpress.com/2015/02/markov01.gif)
An example Markov chain; image source http://www.mathcs.emory.edu/~cheung/ (http://www.mathcs.emory.edu/~cheung/)

In statistics land, they take the "state" interpretation of a random walk very seriously. They call the edge probabilities "state-to-state transitions."

The main theorem we need to do anything useful with Markov chains is the stationary distribution theorem (sometimes called the "Fundamental Theorem of Markov Chains," and for good reason). What it says intuitively is that for a very long random walk, the probability that you end at some vertex v is independent of where you started! All of these probabilities taken together is called the *stationary distribution* of the random walk, and it is uniquely determined by the Markov chain.

However, for the reasons we stated above ("if possible"), the stationary distribution theorem is not true of every Markov chain. The main property we need is that the graph G is *strongly connected.* Recall that a directed graph is called connected if, when you ignore direction, there is a path from every vertex to every other vertex. It is called *strongly connected* if you still get paths everywhere when considering direction. If we additionally require the stupid edge-case-catcher that no edge can have zero probability, then strong connectivity (of one component of a graph) is equivalent to the following property:

For every vertex $v \in V(G)$, an infinite random walk started at v will return to v with probability 1.

In fact it will return infinitely often. This property is called the *persistence* of the state v by statisticians. I dislike this term because it appears to describe a property of a vertex, when to me it describes a property of the connected component containing that vertex. In any case, since in Markov Chain Monte Carlo we'll be picking the graph to walk on (spoiler!) we will ensure the graph is strongly connected by design.

Finally, in order to describe the stationary distribution in a more familiar manner (using linear algebra), we will write the transition probabilities as a matrix A where entry $a_{j,i} = p_{(i,j)}$ if there is an edge $(i, j) \in E$ and zero otherwise. Here the rows and columns correspond to vertices of G, and each *column* i forms the probability distribution of going from state i to some other state in one step of the random walk. Note A is the transpose of the weighted adjacency matrix of the directed weighted graph G where the weights are the transition probabilities (the reason I do it this way is because matrix-vector multiplication will have the matrix on the left instead of the right; see below).

This matrix allows me to describe things nicely using the language of linear algebra. In particular if you give me a basis vector e_i interpreted as "the random walk currently at vertex i," then Ae_i gives a vector whose j-th coordinate is the probability that the random walk would be at vertex j after one more step in the random walk. Likewise, if you give me a probability distribution q over the vertices, then Aq gives a probability vector interpreted as follows:

If a random walk is in state i with probability q_i, then the j-th entry of Aq is the probability that after one more step in the random walk you get to vertex j.

Interpreted this way, the stationary distribution is a probability distribution π such that $A\pi = \pi$, in other words π is an eigenvector of A with eigenvalue 1.

A quick side note for avid readers of this blog: this analysis of a random walk is exactly what we did back in the early days of this blog when we studied the PageRank algorithm (https://jeremykun.com/2011/06/12/googles-pagerank-introduction/) for ranking webpages. There we called the matrix A "a web matrix," did random walks on it, and found a special eigenvalue whose eigenvector was a "stationary distribution" that we used to rank web pages (this used something called the Perron-Frobenius theorem (https://en.wikipedia.org/wiki/Perron%E2%80%93Frobenius_theorem), which says a random-walk matrix has that special eigenvector). There we described an algorithm to actually find that eigenvector by iteratively multiplying A. The following theorem is essentially a variant of this algorithm but works under weaker conditions; for the web matrix we added additional "fake" edges that give the needed stronger conditions.

Theorem: Let G be a strongly connected graph with associated edge probabilities $\{p_e\}_e \in E$ forming a Markov chain. For a probability vector x_0, define $x_{t+1} = Ax_t$ for all $t \geq 1$, and let v_t be the long-term average $v_t = \frac{1}{t}\sum_{s=1}^{t} x_s$. Then:

1. There is a unique probability vector π with $A\pi = \pi$.
2. For all x_0, the limit $\lim_{t \to \infty} v_t = \pi$.

Proof. Since v_t is a probability vector we just want to show that $|Av_t - v_t| \to 0$ as $t \to \infty$. Indeed, we can expand this quantity as

$$\begin{aligned} Av_t - v_t &= \frac{1}{t}(Ax_0 + Ax_1 + \dots + Ax_{t-1}) - \frac{1}{t}(x_0 + \dots + x_{t-1}) \\ &= \frac{1}{t}(x_t - x_0) \end{aligned}$$

But x_t, x_0 are unit vectors, so their difference is at most 2, meaning $|Av_t - v_t| \leq \frac{2}{t} \to 0$. Now it's clear that this does not depend on v_0. For uniqueness we will cop out and appeal to the Perron-Frobenius theorem that says any matrix of this form has a unique such (normalized) eigenvector.

$\square$

One additional remark is that, in addition to computing the stationary distribution by actually computing this average or using an eigensolver, one can analytically solve for it as the inverse of a particular matrix. Define $B = A - I_n$, where I_n is the $n \times n$ identity matrix. Let C be B with a row of ones appended to the bottom and the topmost row removed. Then one can show (quite opaquely) that the last column of C^{-1} is π. We leave this as an exercise to the reader, because I'm pretty sure nobody uses this method in practice.

One final remark is about why we need to take an average over all our x_t in the theorem above. There is an extra technical condition one can add to strong connectivity, called *aperiodicity*, which allows one to beef up the theorem so that x_t itself converges to the stationary distribution. Rigorously, aperiodicity is the property that, regardless of where you start your random walk, after some sufficiently large number of steps n the random walk has a positive probability of being at every vertex at every subsequent step. As an example of a graph where aperiodicity fails: an undirected cycle on an even number of vertices. In that case there will only be a positive probability of being at certain vertices every *other* step, and averaging those two long term sequences gives the actual stationary distribution.

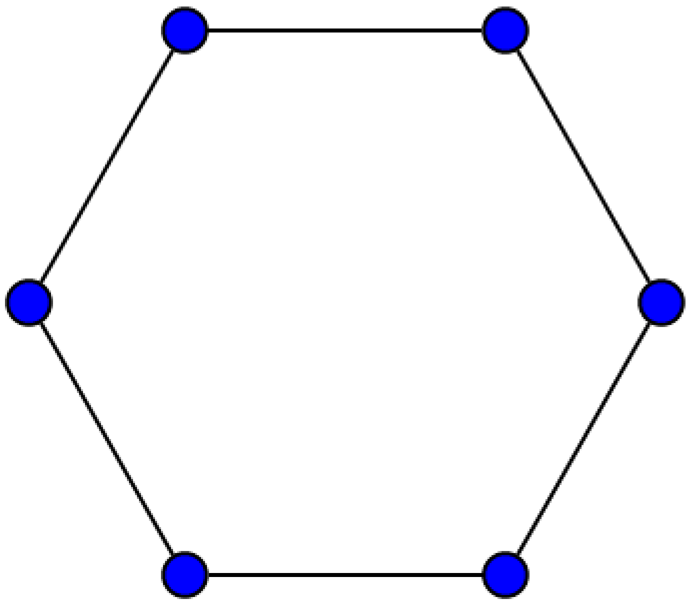


Image source: Wikipedia

One way to guarantee that your Markov chain is aperiodic is to ensure there is a positive probability of staying at any vertex. I.e., that your graph has a self-loop. This is what we'll do in the next section.

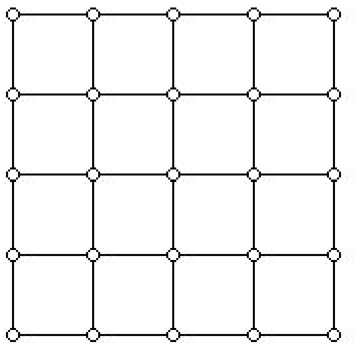
Constructing a graph to walk on

Recall that the problem we're trying to solve is to draw from a distribution over a finite set X with probability function $p(x)$. The MCMC method is to construct a Markov chain whose stationary distribution is exactly p, even when you just have black-box access to evaluating p. That is, you (implicitly) pick a graph G and (implicitly) choose transition probabilities for the edges to make the stationary distribution p. Then you take a long enough random walk on G and output the x corresponding to whatever state you land on.

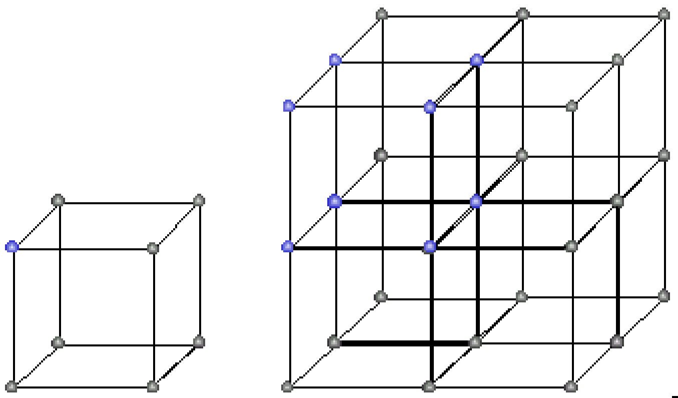
The easy part is coming up with a graph that has the right stationary distribution (in fact, "most" graphs will work). The hard part is to come up with a graph where you can prove that the convergence of a random walk to the stationary distribution is fast in comparison to the size of X. Such a proof is beyond the scope of this post, but the "right" choice of a graph is not hard to understand.

The one we'll pick for this post is called the **Metropolis-Hastings** algorithm. The input is your black-box access to $p(x)$, and the output is a set of rules that implicitly define a random walk on a graph whose vertex set is X.

It works as follows: you pick some way to put X on a lattice, so that each state corresponds to some vector in $\{0, 1, \dots, n\}^d$. Then you add (two-way directed) edges to all neighboring lattice points. For $n = 5, d = 2$ it would look like this:

 (https://jeremykun.files.wordpress.com/2015/02/2dlattice.jpg)
Image credit http://www.ams.org/samplings/feature-column/fcarc-taxi (http://www.ams.org/samplings/feature-column/fcarc-taxi)

And for $d = 3, n \in \{2, 3\}$ it would look like this:

 (https://jeremykun.files.wordpress.com/2015/02/lattice.gif)
Image credit http://www.chem.latech.edu/~upali/ (http://www.chem.latech.edu/~upali/)

You have to be careful here to ensure the vertices you choose for X are not disconnected, but in many applications X is naturally already a lattice.

Now we have to describe the transition probabilities. Let r be the maximum degree of a vertex in this lattice ($r = 2d$). Suppose we're at vertex i and we want to know where to go next. We do the following:

1. Pick neighbor j with probability $1/r$ (there is some chance to stay at i).
2. If you picked neighbor j and $p(j) \geq p(i)$ then deterministically go to j.
3. Otherwise, $p(j) < p(i)$, and you go to j with probability $p(j)/p(i)$.

We can state the probability weight $p_{i,j}$ on edge (i, j) more compactly as

$$p_{i,j} = \frac{1}{r} \min(1, p(j)/p(i))$$
$$p_{i,i} = 1 - \sum_{(i,j) \in E(G); j \neq i} p_{i,j}$$

It is easy to check that this is indeed a probability distribution for each vertex i. So we just have to show that $p(x)$ is the stationary distribution for this random walk.

Here's a fact to do that: if a probability distribution v with entries $v(x)$ for each $x \in X$ has the property that $v(x)p_{x,y} = v(y)p_{y,x}$ for all $x, y \in X$, the v is the stationary distribution. To prove it, fix x and take the sum of both sides of that equation over all y. The result is exactly the equation $v(x) = \sum_y v(y)p_{y,x}$, which is the same as $v = Av$. Since the stationary distribution is the unique vector satisfying this equation, v has to be it.

Doing this with out chosen $p(i)$ is easy, since $p(i)p_{i,j}$ and $p(i)p_{j,i}$ are both equal to $\frac{1}{r}\min(p(i), p(j))$ by applying a tiny bit of algebra to the definition. So we're done! One can just randomly walk according to these probabilities and get a sample.

Last words

The last thing I want to say about MCMC is to show that you can estimate the expected value of a function $\mathbb{E}(f)$ simultaneously while random-walking through your Metropolis-Hastings graph (or any graph whose stationary distribution is $p(x)$). By definition the expected value of f is $\sum_x f(x)p(x)$.

Now what we can do is compute the average value of $f(x)$ just among those states we've visited during our random walk. With a little bit of extra work you can show that this quantity will converge to the true expected value of f at about the same time that the random walk converges to the stationary distribution. (Here the "about" means we're off by a constant factor depending on f). In order to prove this you need some extra tools I'm too lazy to write about in this post, but the point is that it works.

The reason I did not start by describing MCMC in terms of estimating the expected value of a function is because the core problem is a sampling problem. Moreover, there are many applications of MCMC that need nothing more than a sample. For example, MCMC can be used to estimate the volume of an arbitrary (maybe high dimensional) convex set. See these lecture notes (http://www.cs.berkeley.edu/~sinclair/cs294/n1.pdf) of Alistair Sinclair for more.

If demand is popular enough, I could implement the Metropolis-Hastings algorithm in code (it wouldn't be industry-strength, but perhaps illuminating? I'm not so sure…).

Until next time!

This entry was posted in Algorithms, Graph Theory, Linear Algebra, Probability Theory and tagged markov chain, mathematics, MCMC, monte carlo, random walk. Bookmark the permalink.

42 thoughts on "Markov Chain Monte Carlo Without all the Bullshit"

Ben Buckley

April 6, 2015 at 3:13 pm • Reply

It's not immediately obvious to me how this helps with our baby name blackbox. I assume I'm missing something important.

My understanding is that, in the graph, each state would correspond to some name, where n = 26 (letters in the alphabet) and d = 7 (just to keep things simple) so that "MALCOLM" is one of the states. Won't the state's neighbours be crazy strings like "JALCOLM" and "MALCZLM" for which the blackbox should return zero, and p(j)/p(i) is always zero? So, if I do a walk on the graph, how am I supposed to leave the state "MALCOLM"?

1. ◦

   j2kun

   April 6, 2015 at 3:27 pm • Reply

   This is a good observation, and you're right. Unfortunately this is a problem dependant issue, and in particular for names there is nothing stopping someone from making up a name like Jalcom. So the issue is finding a way to map names to grid vertices in a sensible way. I don't know of a simple way to do that off the top of my head without a given enumeration of all legal names.

   ◦

   paulie

   August 30, 2017 at 7:57 am • Reply

   Thank you and God bless you for the inspiration we named our sweet baby boy Jalcolm

   ZL

   April 6, 2015 at 4:23 pm • Reply

   "If demand is popular enough, I could implement the Metropolis-Hastings algorithm in code" yes please

2. ◦

   Hugle (@wulong3)

   April 10, 2015 at 6:12 am • Reply

   There is a working version in python..http://python4mpia.github.io/fitting_data/Metropolis-Hastings.html

3. **gt**

   April 6, 2015 at 11:51 pm • Reply

   Strictly speaking your theorem also requires the state space to be finite: a simple M/M/1 "exploding queue" will serve as a counter example. Having said that, your original motivation was MCMC on a finite state set X, so perhaps this is implicit.

   Amnon Harel

   April 7, 2015 at 3:18 am • Reply

   That's quite a straw man, in the introduction. Not only full of jargon but after two badly chosen and inexact sentences it moves on to specific usages and implementations that do not belong in an introduction, without spelling out, e.g. "what is a Monte Carlo?". This web page gives a very nice introduction to Markov Chain sampling. But the title is Markov Chain Monte Carlo, and all the basic concepts of the Monte Carlo method are missing. To be sure, they are readily available elsewhere: http://en.wikipedia.org/wiki/Monte_Carlo_method

   Still, I would get rid of "expectation values". "Integration" is more accurate, basic, general, and communicative to everyone who went through a calculus course and encountered the fact that sometimes integrals are hard.

4. ◦

   j2kun

   April 7, 2015 at 8:56 am • Reply

   I'm not sure what to say. Any search for "Markov chain sampling method" gives you results for MCMC, or a scientific paper about dirichlet distributions. And the core of any Monte Carlo method is sampling, regardless of whether you use the sample to estimate an integral.

5. **Richard**

   April 7, 2015 at 7:53 am • Reply

   Your explanation is great. Thanks for writing this.

One small comment, though: In your definition of the sampling problem you use f both as a probability density function and as a random variable, and that was a little confusing. It would be very helpful (at least for me) if you used different symbols here (assuming they are meant to be different?).

Josh

April 7, 2015 at 8:16 pm • Reply

Nice post! And I've found many of your other primers very helpful as well.

Quick question: you wrote that a markov chain is essentially a random walk on a graph. In many important situations, however, we define markov chains on continuous state spaces, and I'm not sure I see how that fits into the framework you described. Can markov chains on continuous state spaces be interpreted as random walks on (implicit) graphs?

Also, a perhaps clearer introduction to MCMC than the one you cited is in chapter 29 of David MacKay's book: http://www.inference.phy.cam.ac.uk/itprnn/book.html

6. ◦ **j2kun**

   April 7, 2015 at 8:34 pm • Reply

   You can define graphs on continuous state spaces just fine, and just as you would for a usual Markov chain you can define the transitions implicitly and talk about densities as integrals, etc.

   ◦ **j2kun**

   April 7, 2015 at 8:36 pm • Reply

   And yes that text does appear to have a great treatment of the subject.

Ian Mallett

April 7, 2015 at 10:16 pm • Reply

Very nice article; thanks!
One minor clarification: "you go to j with probability p_j / p_i." |-> "you go to j with probability p(j) / p(i)."

7. ◦ **j2kun**

   April 7, 2015 at 10:39 pm • Reply

   Fixed, thanks!

Tyson Williams

April 14, 2015 at 9:48 pm • Reply

Nice post. I completely agree that most explanations of MCMC are too jargon dense. Your treatment here is great.

Given that your opening example was picking baby names, I was anxiously looking forward to how you were going to define that two name are adjacent in the state graph. I became disappointed when I read "you pick some way to put X on a lattice" 😉

8. ◦ **j2kun**

   April 14, 2015 at 10:59 pm • Reply

   For what it's worth, I'm pretty sure the set of baby names is sparse in the set of all strings, but yeah it's a cop out.

Andreas Eckleder

April 16, 2015 at 2:41 am • Reply

I think from your description it is not immediately clear that your black box does not have a finite vocabulary of names but the name is really an arbitrary string. I think it would make understanding this great article a lot easier if you explicitly mentioned that out at the beginning.

9. ◦ **j2kun**

   April 16, 2015 at 8:33 am • Reply

   It is finite.

Nick

April 20, 2015 at 4:40 pm • Reply

A bit of a thought about the statement that a graph being strongly connected is equivalent to "For every vertex v \in V(G), an infinite random walk started at v will return to v with probability 1."

I can see how the former implies the latter, but without also requiring at least connectedness already, the later does not seem to imply the former. Consider a graph with more than one vertex, where each vertex has exactly one edge which connects to itself with probability 1. It definitely satisfies the latter property, but is also not strongly connected, or even connected at all.

The property "for every pair of vertices u,v \in V(G), and infinite random walk started at u will pass through v with probability 1." would imply strong connectedness, and I think, though I haven't worked out the proof, that strong connectedness and all edge probabilities positive would imply it.

Am I going horribly wrong here? Is equivalent only being used as a one directional implication rather than as an iff?

10. ◦ **j2kun**

    April 20, 2015 at 9:02 pm • Reply

    You're right, I was being unclear. The confusion is because what the statistics community calls "persistent" (which is the definition you quoted) really means "the connected component containing a vertex is strongly connected." It's sort of silly because for any Markov chain you assume you're working with a connected graph (in which case persistent means the whole graph is strongly connected), because to analyze a graph which is a union of connected components you just analyze the connected components one by one. I have updated the text to reflect this.

11. **Lorand**

    April 24, 2015 at 4:07 am • Reply

    Hi, thanks for the article!
    It is not clear to me how to choose the transition probabilities from one state (name) to another state.
    Also, is it important which neighbors a state has, or can i just randomly assign names to verices in the lattice (let's say i would have 100 names and assign them randomly to a lattice of 10×10)?

isomorphismes

April 26, 2015 at 12:44 am • Reply

Agree. Statisticians often let notation get the better of them.

12. ◦ **isomorphismes**

    April 26, 2015 at 12:45 am • Reply

    *notation and verbiage

13.

Evan

June 20, 2015 at 3:45 pm • Reply

In the last paragraph of the “Constructing a Graph to Walk On” section, I think there’s a small error in this sentence:

“Doing this with out chosen p(i) is easy, since p(i)p_{i,j} and p(i)p_{j,i} are both equal to \frac1r \min(p(i), p(j)) by applying a tiny bit of algebra to the definition.”

I think “p(i)p_{i,j} and p(i)p_{j,i}” is meant to be “p(i)p_{i,j} and p(j)p_{j,i}” (note the “i” replaced by “j” as the argument to the second p()).

Thanks for the great article!

14.
asmageddon

June 27, 2015 at 3:23 pm • Reply

> without all the bullshit
> mathematical notation everywhere
No thanks.

15.
Samchappelle

August 6, 2015 at 11:23 pm • Reply

Reblogged this on schapshow.

16.
Tann

January 12, 2016 at 3:37 am • Reply

Glad I didn’t get the version *with* all the bullshit. This is hard enough

Navaneethan

March 17, 2016 at 12:46 pm • Reply

I had a question about MCMC that I’m finding hard to answer. If the idea is to sample from a complicated distribution, how do you know that you’ve produced a representative sample? Is there a property of MCMC that ensures that the sample is representative? If not, isn’t that a huge weakness of this framework?

17.
- **j2kun**

  March 17, 2016 at 1:13 pm • Reply

  Yes, in fact that is the entire point, and and I gave a mathematical proof that it works in the post.

18.
mariusagm

August 12, 2016 at 9:55 pm • Reply

Reblogged this on Marius.

19.
thweealc

September 20, 2016 at 1:02 am • Reply

should (u,v) element E correctly be (u,v) element V?

20.
Cindy Yeh

September 24, 2016 at 1:41 pm • Reply

Thank you, I needed help to understand what a MCMC is, after listening to Ed Vul's talk about cognitive biases and trying to model how reasoning works and seeing what biases it explains. I thought it was a very interesting talk, here https://www.youtube.com/watch?v=eSq_80TfUO0 Plan to read more of your blog posts. Thanks very much.

21.

Boris Jensen

April 6, 2017 at 5:35 am • Reply

Thanks for the nice writeup.
— Doing this with out chosen p(i) is easy, since p(i)p_{i,j} and p(i)p_{j,i} are both equal to \frac1r \min(p(i), p(j))
Don't you mean p(i)p_{i,j} and p(j)p_{j,i}? Or am I misunderstanding something?

22.

Richard

April 19, 2017 at 3:46 pm • Reply

It is more general than this right? The black box is some constant k times p(x)?
So you only need to know the proportions of the probabilities via the black box

compostbox

May 4, 2017 at 9:37 pm • Reply

Thanks for the good writing.
However, there is one thing not obvious to me. Why the lattice thing is necessary at all in Metropolis-Hastings? Imagine a fully linked graph where you can move to any state from any state. It seems to me everything will still work. Put it another way, it's just that the r will now equal to the number of size of X minus 1, and notice in your proof the exact value of r does not matter.

23. ◦

j2kun

May 4, 2017 at 10:06 pm • Reply

This is correct, however, often the size of X is exponentially large, and so a complete graph will not be tractable.

For example, suppose your state space is the integer grid . You may want your algorithm to run in time polynomial in , but there are states. This is why I brought up the example of names, since it is also an exponentially large state space.

allenhw

May 8, 2017 at 11:43 am • Reply

I first wanted to thank you and great job for getting ideas across clearly. It was super helpful!

I have one question; what's the advantage of MCMC over simpler algorithms using RNG? For example, you can simulate a dice roll by dividing [0.1] into 6 subspaces, generating a random number x in [0,1], and outputting result based which subspace x falls into. This can be done as long as we have p(x) for all X.

24. ◦

j2kun

May 8, 2017 at 12:27 pm • Reply

MCMC is used when the number of possible outputs is exponentially large. To see this, imagine your proposed die had 2^50 sides, each of which had a slightly different probability of occurring (and there is no discernible pattern or formula to tell you p(x) for a given x, you just have to ask the black box p(x) to give you a value). How does your algorithm break down in this scenario? How long does it take to simulate a single die roll on average?

25.

Rob Forgione

June 6, 2017 at 11:31 am • Reply

Thanks for writing this! Do you mind explaining why x_t and x_0 are unit vectors in your proof of the eigenvector theorem? I was under the impression that each x vector is a probability vector, which means it would sum to 1 but not necessarily have length 1. Any help/clarification is appreciated — thanks!

Matt

June 15, 2017 at 7:47 am • Reply

Thanks for writing this. It was very helpful to understand the basic idea. Unfortunately, I still have some troubles to understand the last step (Constructing a graph to walk on). Could you maybe explain on a simple example how you build up the correct graph?

Maybe for the following example:

Let us assume
X = {A,B,C}
with
p(A) = 2/10, p(B) = 5/10, p(C) = 3/10.
This means our stationary distribution should be
p = [2/10, 5/10, 3/10] (correct me if I am wrong).

How would you now build up the correct Graph?
What would in this case be n and d you used to build the lattice?
How would the lattice look like?

Thanks for your answers 🙂

26. ◦

j2kun

June 15, 2017 at 8:15 am • Reply

In your example, the number of nodes in the graph is only 3, which is so small that any connected graph should work, if I'm not mistaken.

In general, there is no correct answer. You want a graph which is sparse, but also has high connectivity properties, meaning you need to remove many edges in order to disconnect the graph. Graphs that contain only one or two shortest paths between two nodes would have bad mixing. Grid graphs tend to do well, but I don't know how to pick the parameters in a principled way. There is also a theory of expander graphs that is closely related, and you may want to look up that literature.

◦

Matt

June 15, 2017 at 8:39 am

Maybe I am little bit more confused than I thought. Have I understand the procedure correct?

We need to find a Graph that has the stationary distribution
p = [2/10, 5/10, 3/10]
This is the same like saying: We need to find a Matrix A with Eigenvalue 1 and Eigenvector p.
If we have this we could walk on the Graph represented by matrix A. If we do this a long time the vertex we will end could be used as generator for random variables probability p.
Is this correct?