Short-Cut MCMC: An Alternative to Adaptation

Radford M. Neal

Dept. of Statistics and Dept. of Computer Science
University of Toronto

http://www.cs.utoronto.ca/∼radford/

Third Workshop on Monte Carlo Methods, Harvard, 13-14 May 2007

Outline of This Talk

- Tuning of MCMC methods, and the adaptation approach to doing it.
- The idea of short-cut MCMC, described informally.
- Interlude: Two ideas needed to formalize short-cut MCMC:
  - temporarily mapping to another space
  - caching previous calculations
- Short-cut MCMC, the formal version.
- Practical aspects and limitations of short-cut MCMC.

The Need to Tune MCMC Methods

Many MCMC methods have parameters that have to be tuned —eg, the width of the proposal distribution for Metropolis updates. We seldom have reliable *a priori* knowledge of the right values for these tuning parameters.

Such parameters are often tuned using preliminary MCMC runs, then kept fixed during the final runs. This works fairly well, but a better method would be nice.

Many people have investigated methods for adapting the tuning parameters *during* a run, based on statistics collected earlier in the run. Unfortunately, such adaptation undermines the Markov property needed to prove that the chain converges to the desired distribution, and in general it doesn't.

Eventual convergence can be guaranteed if adaptation becomes slower and slower with time. But does this have any real advantage over just fixing the parameters after a preliminary period of adaptation?

I propose another idea, which doesn't disturb the Markov property. It also allows different parameter values to (in effect) be used in different regions.

Short-Cut MCMC: Adapting Without Really Adapting

Rather than adapt a tuning parameter (say σ), we can just cycle through m values of the parameter, $\sigma_1, \ldots, \sigma_m$, performing N_i updates using σ_i.

This way, we may sometimes use a good value of σ. Unfortunately, we also waste computation time using bad values of σ.

The idea: Somehow arrange that the amount of computation time devoted to the bad values of σ is small, while still, in a mathematical sense, performing the pre-determined number of updates.

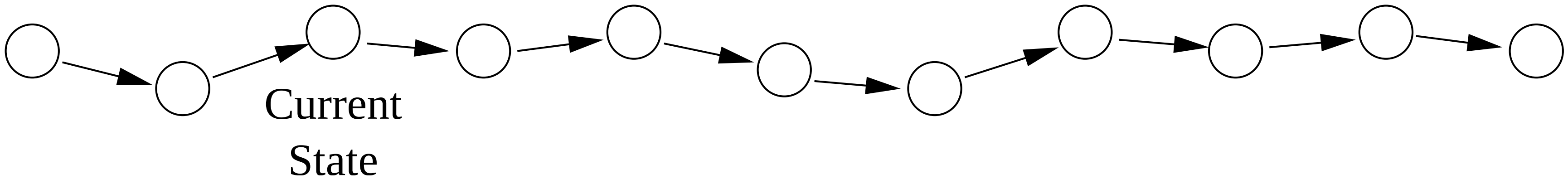
Since we don't adapt in a mathematical sense, the convergence proofs still apply.

Since we do adapt in a computational sense, we spend most of our computation time on updates that use a good value of σ.

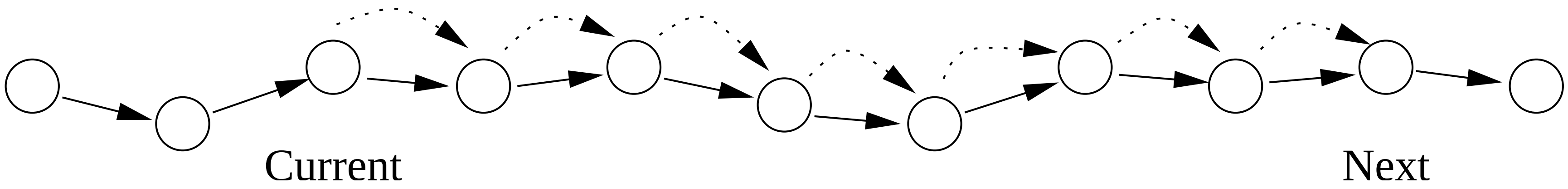
But how can one arrange this?

An Informal Picture of How the 'Short-Cut' Method Works

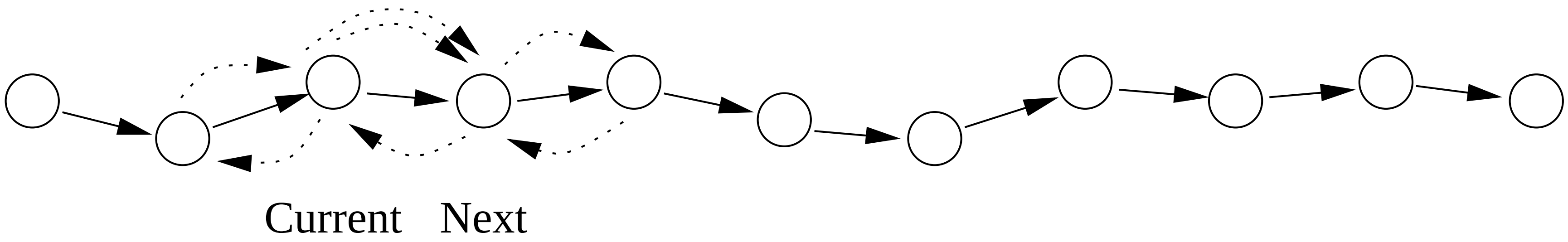
Imagine simulating Markov transitions, with fixed tuning parameters, from the current state indefinitely far forward, and also indefinitely far backward (using reversed transitions). This produces an infinite chain of states:



N transitions done the ordinary way move us N steps forward in this chain:



Rather than continuing forward for N steps, suppose we sometimes reverse — specifically, whenever the state indicates that the tuning parameters aren't good:



After two reversals, we just go back and forth over the same states, and needn't do any more computations, provided we cached the previously-visited states.

Interlude:

MCMC methods that use temporary mapping and caching

Notation

We wish to sample from some distribution for $x \in \mathcal{X}$ that has probability mass or probability density function $\pi(x)$.

We do this by simulating a Markov chain with transition probabilities/densities (from x to x') denoted by $T(x'|x)$, for which π is an invariant distribution:

$$\int \pi(x)\, T(x'|x)\, dx \quad = \quad \pi(x')$$

We can define reversed transitions as follows:

$$\tilde{T}(x'|x) \quad = \quad T(x|x')\pi(x')/\pi(x)$$

These also leave π invariant. For "reversible" transitions, $\tilde{T} = T$.

Transitions that Temporarily Map to Another Space

One way to define the transitions $T(x'|x)$ is via three other stochastic mappings, $\hat{T}$, $\bar{T}$, and $\check{T}$, as follows:

$$x \quad \xrightarrow{\hat{T}} \quad y \quad \xrightarrow{\bar{T}} \quad y' \quad \xrightarrow{\check{T}} \quad x'$$

Starting from x, we obtain a value in a temporary state space, $\mathcal{Y}$, by sampling from $\hat{T}(y|x)$. The target distribution for y has probability/density function $\rho(y)$. We require that

$$\int \pi(x)\,\hat{T}(y|x)\,dx \quad = \quad \rho(y)$$

$\bar{T}(y'|y)$ defines a transition on the temporary space that leaves ρ invariant:

$$\int \rho(y)\,\bar{T}(y'|y)\,dy \quad = \quad \rho(y')$$

Finally, $\check{T}$ gets us back to the original space. It must satisfy

$$\int \rho(y')\,\check{T}(x'|y')\,dy' \quad = \quad \pi(x')$$

The overall transition, $T(x'|x)$, leaves π invariant, and so can be used for Markov chain sampling from π.

Methods that Use Temporary Mapping

Many previous and new methods can be viewed as doing this:

- Temporarily mapping to a bigger space $\mathcal{Y} = \mathcal{X} \otimes \mathcal{Z}$, in which an auxiliary variable, z, is introduced — eg, methods using Hamiltonian dynamics, and cluster methods like Swendsen-Wang.

- Temporarily mapping to a smaller space in which a component of x is marginalized away.

- MCMC forms of the Nelder-Mead simplex method for optimization seem possible, in which $\mathcal{Y} = \mathcal{X}^{n+1}$.

- Temporary discretization — eg, as done with embedded Hidden Markov models.

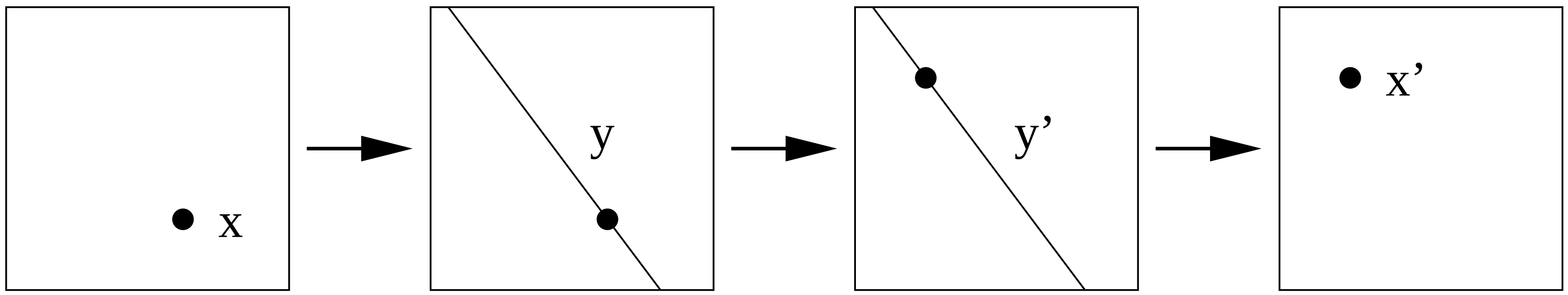
Temporary discretization involves a particular type of mapping that's formally to a bigger space, but effectively to a smaller one. Here's another example...

Hit and Run: Mapping to a Bigger But Effectively Smaller Space

We can map to a bigger space, but then choose $\bar{T}$ to be **non-ergodic**, so that at any particular time, the space we map to is effectively smaller. ‘Hit and run’ methods can be seen in this light.

Here, $\mathcal{X}$ is n-dimensional Euclidean space; $\mathcal{Y}$ is the space of lines in $\mathcal{X}$ together with a marked point on the line.

$\hat{T}$ randomly picks a line passing through x, with all directions equally likely, and makes x the marked point. $\check{T}$ sets x to the marked point on the line, then forgets the direction of the line. $\bar{T}$ moves the mark on the line, not changing the line itself.



Although $\bar{T}$ is non-ergodic, the full transition, T, in which the line direction is chosen randomly, may well be ergodic.

Caching Probability Evaluations for Re-use

Many transitions (eg, Metropolis updates) require evaluation of $\pi(x)$, up to some possibly-unknown normalization factor, for the current point and candidate points.

We can save ('cache') computed values of $\pi(x)$, and re-use them when needed.

Simple contexts where this can be beneficial:

- If we reject a Metropolis proposal, the next x will be the same as the current x. Saving the value of $\pi(x)$ avoids recomputing it.
- If we accept a Metropolis proposal, the next x will be the current candidate state, x^*. Saving $\pi(x^*)$ avoids computing $\pi(x)$ next time.
- Caching the values of $\pi(x^*)$ saves time if we propose the same x^* again.
- Caching values of $\pi(x)$ saves time if we backtrack, so that the x at a new time is the same as the x at some previous time.

The first two situations arise even for continuous spaces, and are handled efficiently by most MCMC implementations.

Mapping to Realizations of a Markov Chain

Let $R(x'|x)$ be the transition probabilities for some inner Markov chain, which leave the distribution $\eta(x)$ invariant. Let $\tilde{R}(x'|x) = R(x|x')\eta(x')/\eta(x)$ be the reverse transition probabilities ($\tilde{R} = R$ for reversible chains).

We map can from $\mathcal{X}$ to the space, $\mathcal{Y}$, of realizations of this inner Markov chain, of length K, with one time step of this chain being 'marked'. The current state, $x \in \mathcal{X}$, is mapped to a state in $\mathcal{Y}$ using a $\hat{T}$ that operates as follows:

1) Choose k uniformly from $0, \ldots, K-1$.
2) Simulate $K-1-k$ forward transitions, R, starting at $x_k = x$, producing states labeled $x_{k+1}, \ldots, x_{K-1}$.
3) Simulate k reverse transitions, $\tilde{R}$, starting at $x_k = x$, labeled $x_{k-1}, \ldots, x_0$.
4) Set the marked time step to k.

$\check{T}$ maps from $\mathcal{Y}$ to $\mathcal{X}$ by just taking the state at the marked time step.

Moving Along the Inner Chain

We can now define a non-ergodic transition, $\bar{T}$, on $\mathcal{Y}$ that only moves the mark, keeping the realization of the inner chain fixed. Effectively, $\bar{T}$ operates on the space $\{0, \ldots, K-1\}$ of positions on the realization of the inner chain.

These transitions must leave ρ invariant, where ρ has been implicitly defined as the distribution resulting from applying $\hat{T}$ to an x drawn from π.

If $\eta = \pi$, it's easy to show that $\bar{T}$ should leave the uniform distribution on $\{0, \ldots, K-1\}$ invariant. More generally, $\bar{T}$ should leave invariant the distribution in which the mark is at time step k with probability proportional to $\pi(x_k) \,/\, \eta(x_k)$.

Note that we don't need to actually compute the whole discretizing chain at the beginning — we can extend it only as needed. We can then let K go to infinity. Note also that we can cache states of the chain that have been computed, so that we don't need to recompute them if we return to that state.

One possibility for $\bar{T}$ is a sequence of M random-walk Metropolis updates. There are many possible applications of such methods — eg, η could be some easy-to-compute part of π, and proposals could make big jumps, potentially moving a large distance with only one evaluation of the costly part of π.

End of Interlude:

Back to short-cut MCMC

Short-Cut MCMC Seen in Terms of a Mapping to Realizations of an Inner Chain

I'll now present the short-cut MCMC method more formally.

We cycle through m settings for the tuning parameter(s), $\sigma_1, \ldots, \sigma_m$. When using σ_i, we will in some sense perform N_i updates given by the transitions T_i (which are defined using the value σ_i).

These N_i updates are actually one transition, defined in terms of a temporary mapping to the space of realizations of an inner chain, with transitions T_i.

The updates on this space just move a marker placed at one position on the inner chain. The updates should leave invariant the uniform distribution for positions of the marker. These inner updates will sometimes be **non-ergodic** — an essential point, since this creates the possibility of the marker staying in a small region, even though N_i is big, in which case only a few transitions will involve actual computation.

Initially, the marker is at the current state. At the end, the new state becomes the one where the marker is located at that time.

Determining When Tuning Parameters are Bad

To use short-cut MCMC, we need to be able to recognize when the tuning parameter(s) — eg, the width of a random-walk Metropolis proposal distribution — are not appropriate.

Sometimes, we might make such a judgement based on just one state from $\mathcal{X}$. For instance, the eigenvalues of the Jacobian matrix of $\log \pi(x)$, evaluated at the current state, might be informative.

More generally, however, we'll want to look at a sequence of J consecutive states from the chain. For Metropolis updates, we could look at the number of rejected transitions in this sequence (or at the sum of the rejection probabilities for rejected transitions). The width of a proposal distribution is inappropriate if the number of rejections is either very high or very low. For this judgement, a sequence length of around $J = 10$ may be sufficient.

For short-cut MCMC to work well, the total number, N_i, of updates done with parameters σ_i should be much bigger than J.

How to Continue in the Same Direction

When σ_i is an appropriate value for the tuning parameter, we want short-cut MCMC to effectively just perform N_i updates using T_i — ie, to move the marker from its initial position on the inner chain to the point N_i steps forward. We don't want the marker to move back and forth along the inner chain in a random walk.

To accomplish this, the marker will be an arrow, pointing one direction or the other along the chain. Formally, we map from the state x to a state $y = (c, k, s)$, where

- c is a realization of an inner chain with x as one of it's states (say x_0)
- k marks a position on the chain
- $s \in \{-1, +1\}$ indicates a direction for moving the marker.

Operation of a Short-Cut MCMC Update

A short-cut update with tuning parameters σ_i operates as follows:

1) Map from the current state, x, to $y = (c, k, s)$. Here, c is a realization (not computed yet) of an inner chain (using transitions T_i) passing through $x = x_0$, so that $k = 0$. We draw s uniformly from $\{-1, +1\}$.

2) Do the following updates for k and s:

   a) Set $k = k + Js$. This moves us J positions along the inner chain. We'll actually compute these J transitions at this point.

   b) Repeat the following $N_i - J$ times:

      - If the transitions between x_{k-Js} and x_k indicate that σ_i is inappropriate, set $k = k - Js$, and then set $s = -s$ (ie, we reverse direction).
      - If the transition from x_k to x_{k+s} hasn't been computed before (in either the forward or the reverse direction), compute it now.
      - Set $k = k + s$.

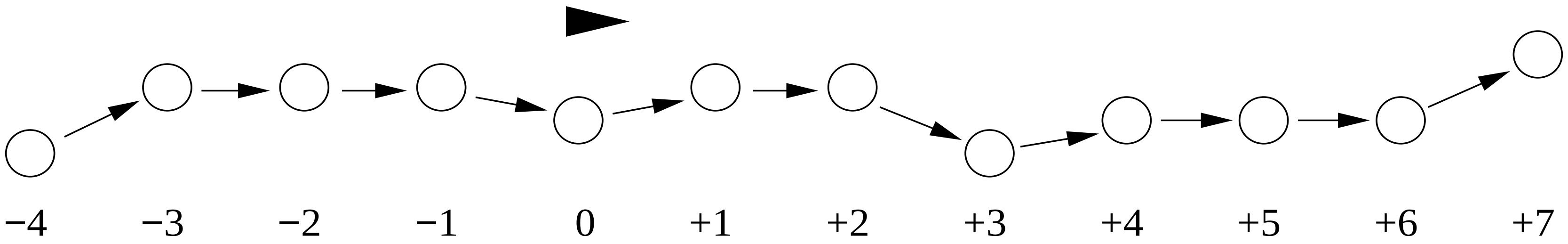
3) Let the new state be $x' = x_k$.

Note that the updates in (2) leave the uniform distribution for k and s invariant. The overall update therefore leaves π invariant.

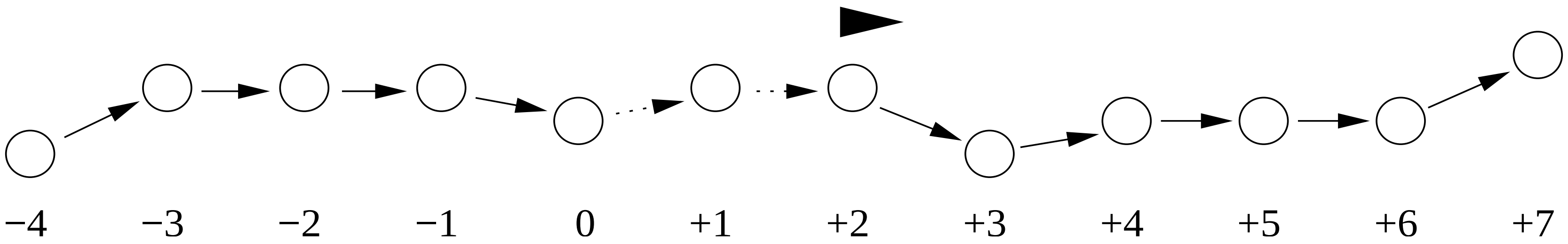
An Illustration, Part I

Here's an illustration of a short-cut MCMC update with $J = 2$. The horizontal axis represents 'time' for the inner Markov chain; the vertical axis represents $\mathcal{X}$.

Here's the initial state for the update:



We start by moving J positions in the direction of the arrow (step 2(a)):

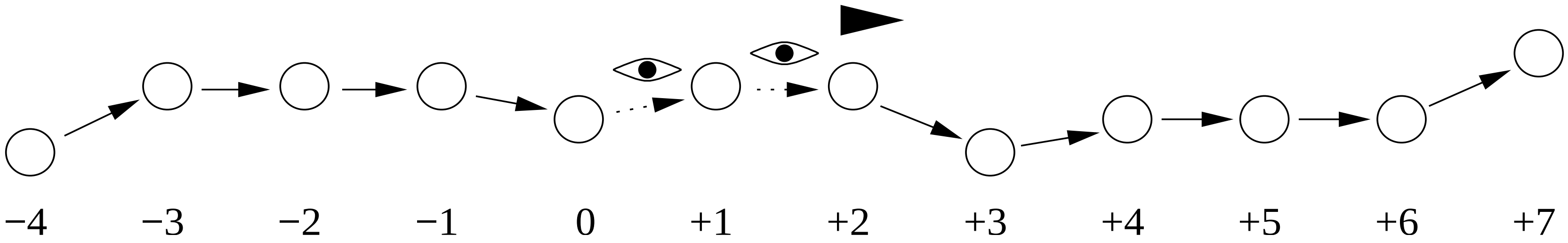


The transitions that have actually been computed are shown as dotted arrows.

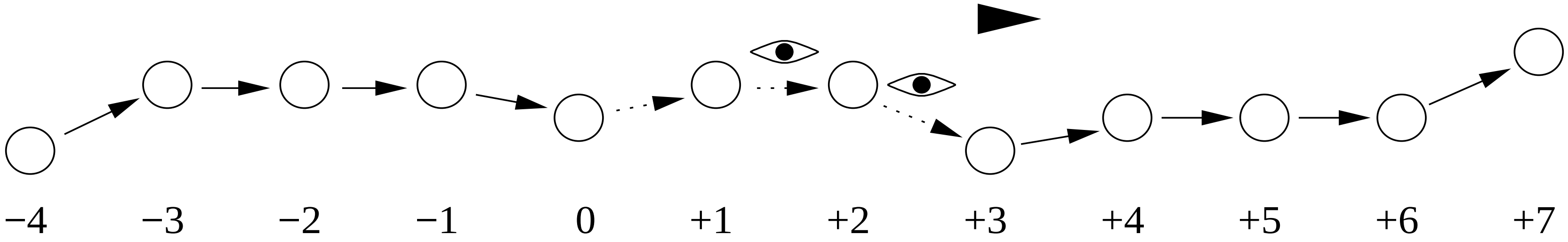
We continue with $N_i - J$ movements as described by step 2(b)...

An Illustration, Part II

We first look at the previous $J = 2$ transitions, to see whether they indicate that σ_i is inappropriate:



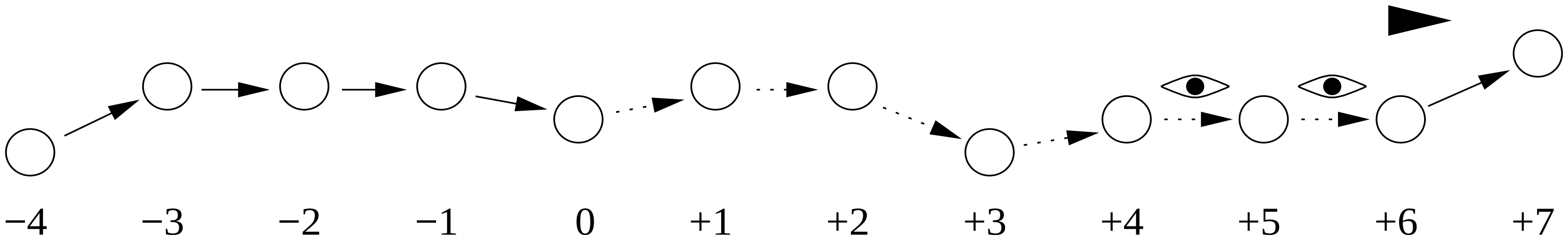
Suppose that we consider σ_i inappropriate only if both previous transitions were rejections. Only one is a rejection here, so we move in the direction of the arrow, and look again:



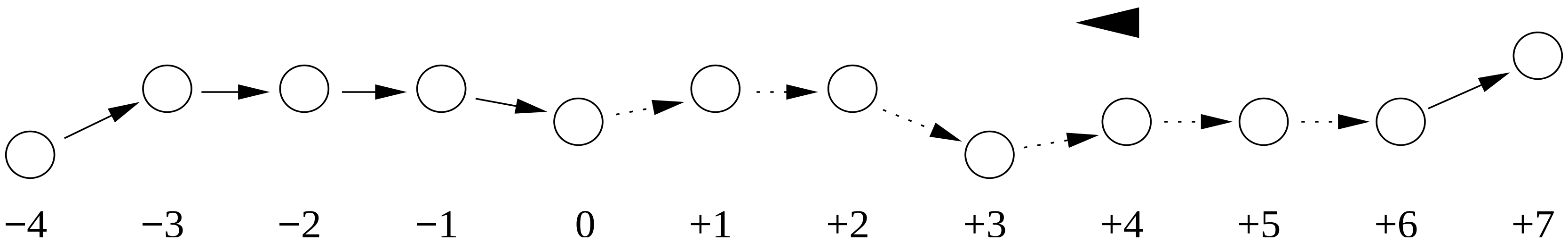
Again, only one of the transitions looked at is a rejection, so we continue in the direction of the arrow...

An Illustration, Part III

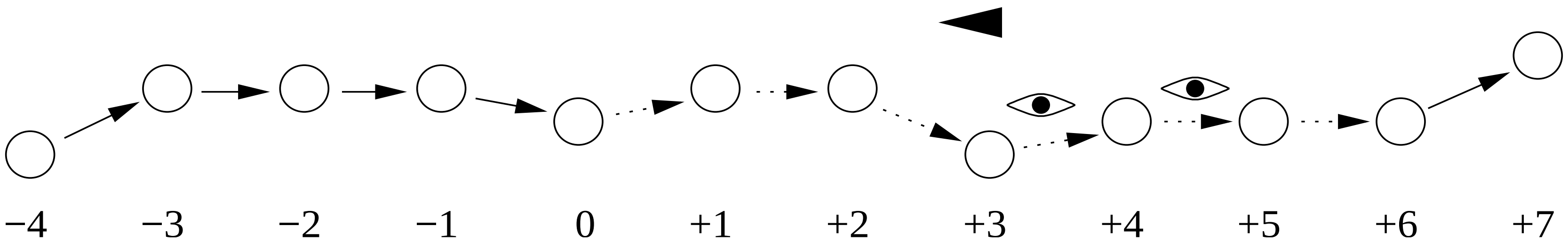
After a few more steps, we get to this situation:



Both transitions looked at here are rejections, so we reverse direction (and also move back J steps):



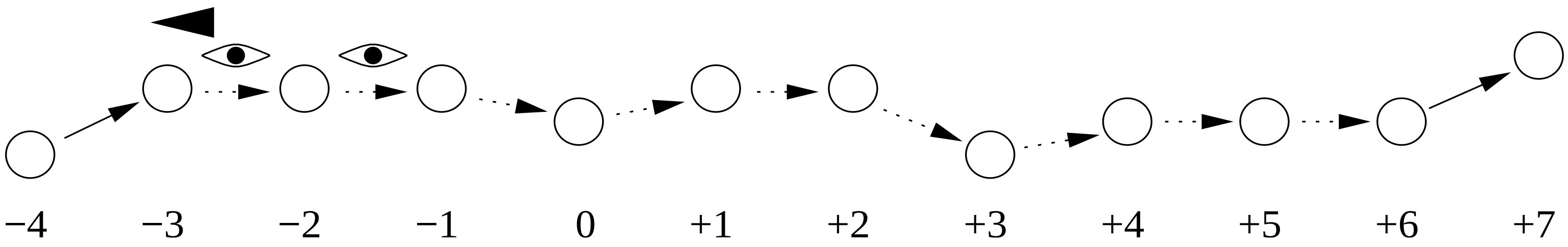
We now move to the left, and again look at the previous two transitions:



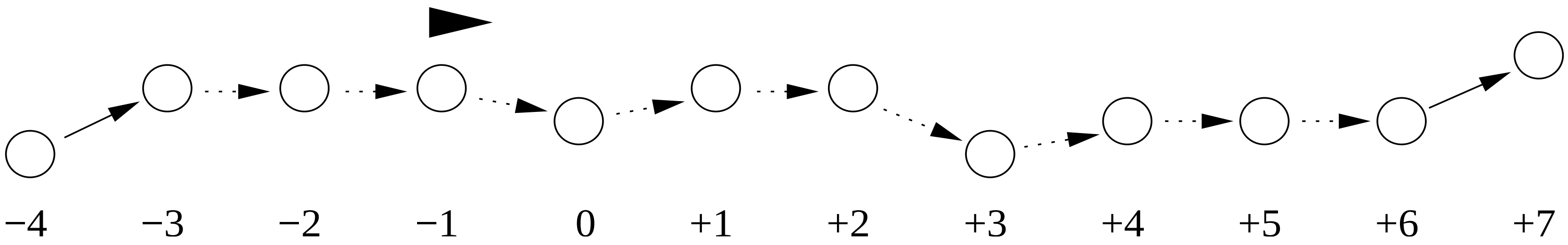
We continue in this way to the left, until...

An Illustration, Part IV

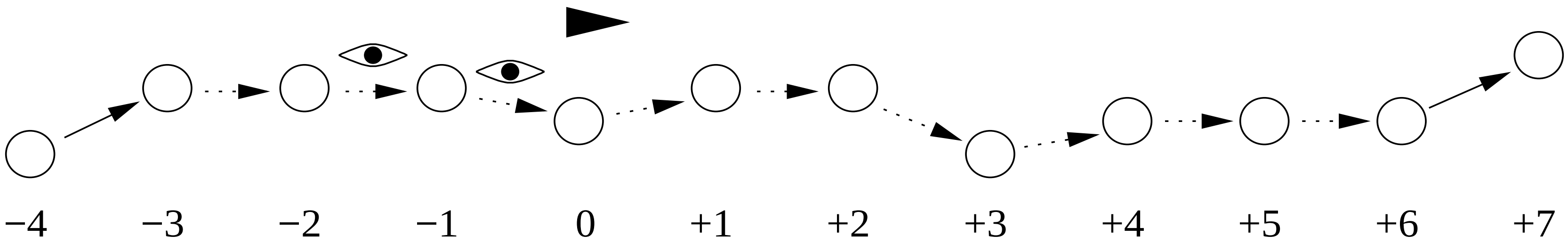
We finally reach a point going left when the previous two transitions are rejections:



We reverse direction again (and move back J steps):



We then move in the arrow direction and look at the previous two transitions:



If N_i is large enough, we will again reverse when we reach position $+6$ on the right, and then again reverse when we reach position -3 on the left, and so forth. But all this requires **no new computation**.

Summary of the Illustration

In this example of a short-cut MCMC update, only nine transitions needed to be computed — *regardless* of how large N_i is.

The final state will be one somewhere between position -3 and position $+6$, including the possibility of position 0, where we started. Little will have been accomplished, but at little cost.

However, for some other value of σ_i, which is more appropriate, we will not reverse (or will reverse only much later), and the end state will be up to N_i steps from the start state. Such an update will require computing up to N_i transitions. A lot will have been accomplished, at a corresponding computational cost.

The effect: Most of our computation time is spent on updates with appropriate values for the tuning parameters.

Some practical issues

For Metropolis updates, we might regard a proposal width, σ, as inappropriate if the rejection rate is either too high (σ too large) or too low (σ too small). With this criterion, we might try $\sigma_1 = 0.1$, $\sigma_2 = 1$, and $\sigma_3 = 10$, all with $N_i = 1000$ and $J = 10$.

Alternatively, we can consider σ to be inappropriate only when it leads to a high rejection rate. We can still avoid spending much time with overly small σ by varying N_i — eg, with the σ_i above, use $N_1 = 250$, $N_2 = 1000$, and $N_3 = 4000$.

Rather than cycling through a small set of σ_i, we could instead sample a (σ_i, N_i) pair from some distribution.

We can tune methods other than Metropolis: Eg, for univariate slice sampling updates, we might look at the ratio of the size of the initial interval to the size of the interval that the accepted point was sampled from.

Limitations of Short-Cut MCMC

A short-cut MCMC update will always evaluate at least $J+1$ transitions. There is also some computational overhead in doing an update.

So the fraction of σ_i values that are appropriate must not be tiny, if short-cut MCMC is to spend most of its time on computations for good values.

One consequence is that we can't use short-cut MCMC to 'adapt' a large number of tuning parameters, if only a tiny part of the full parameter space is appropriate. For example, it won't be feasible to 'tune' a large covariance matrix for Metropolis proposals.

References

All my documents below are available from `http://www.cs.utoronto.ca/~radford/`

R. M. Neal (1993) *Probabilistic Inference Using Markov Chain Monte Carlo Methods.*

General review of Markov chain Monte Carlo methods.

R. M. Neal (2005) "The short-cut Metropolis method".

Introduces the short-cut MCMC method as applied to Metropolis updates, in a specialized way based on expressing Metropolis updates in terms of deterministic operations.

R. M. Neal (2006) "Constructing efficient MCMC methods using temporary mapping and caching", slides for a talk.

Describes the general idea of using temporary mappings and caching to create new MCMC methods, including short-cut MCMC.