DAG–models and Markov Chain Monte Carlo methods – a short overview

Søren Højsgaard

Institute of Genetics and Biotechnology

University of Aarhus

August 6, 2008

Printed: August 6, 2008 File: DAGMC-Lecture.tex

Contents

1 Bayesian modeling and DAG models

The usual Bayesian setting is

$$p(y, \theta) = p(y|\theta)p(\theta)$$

Want to draw inference on the posterior $p(\theta|y)$,

$$p(\theta|y) = \frac{p(y|\theta)p(\theta)}{p(y)}$$

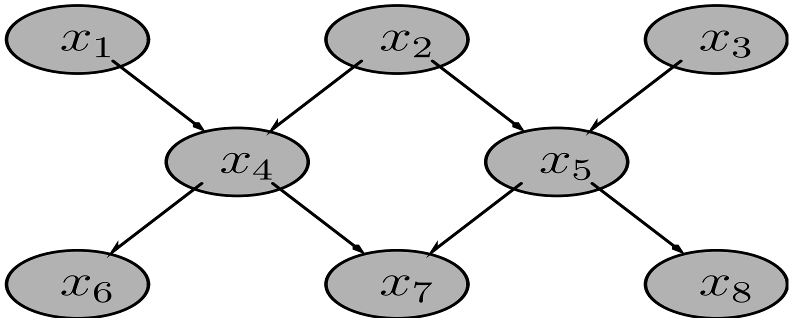
where $p(y) = \int p(y|\theta)p(\theta)d\theta$ is the (usually intractable) normalizing constant.

DAG is short for Directed Acyclic Graph. DAG–models (discussed below) can come into play in two places:

- Model specification: The model $p(y, \theta) = p(y|\theta)p(\theta)$ can be high dimensional.
- Inference: When calculating $p(\theta|y)$

2 DAG–models – model specification

One way of looking at DAG–models is as a way of specifying a (complex) multivariate distribution through smaller “building blocks”, namely univariate conditional distributions.



$$
\begin{aligned}
p(x_1, x_2, \ldots, x_8) &= p(x_1)p(x_2)p(x_3)p(x_4|x_1, x_2)p(x_5|x_2, x_3) \quad (1)\\
& p(x_6|x_4)p(x_7|x_4, x_5)p(x_8|x_5) \quad (2)
\end{aligned}
$$

More generally:

$$
p(X) = \prod_i p_{x_i}(x_i|parents(x_i))
$$

The variables/nodes x_i can be observables, parameters and – when orphan – covariates and hyperparameters.

Key point: The graph being acyclic implies that

$$\prod_i p_{x_i}(x_i | parents(x_i))$$

specifies a valid joint density over the variables X.

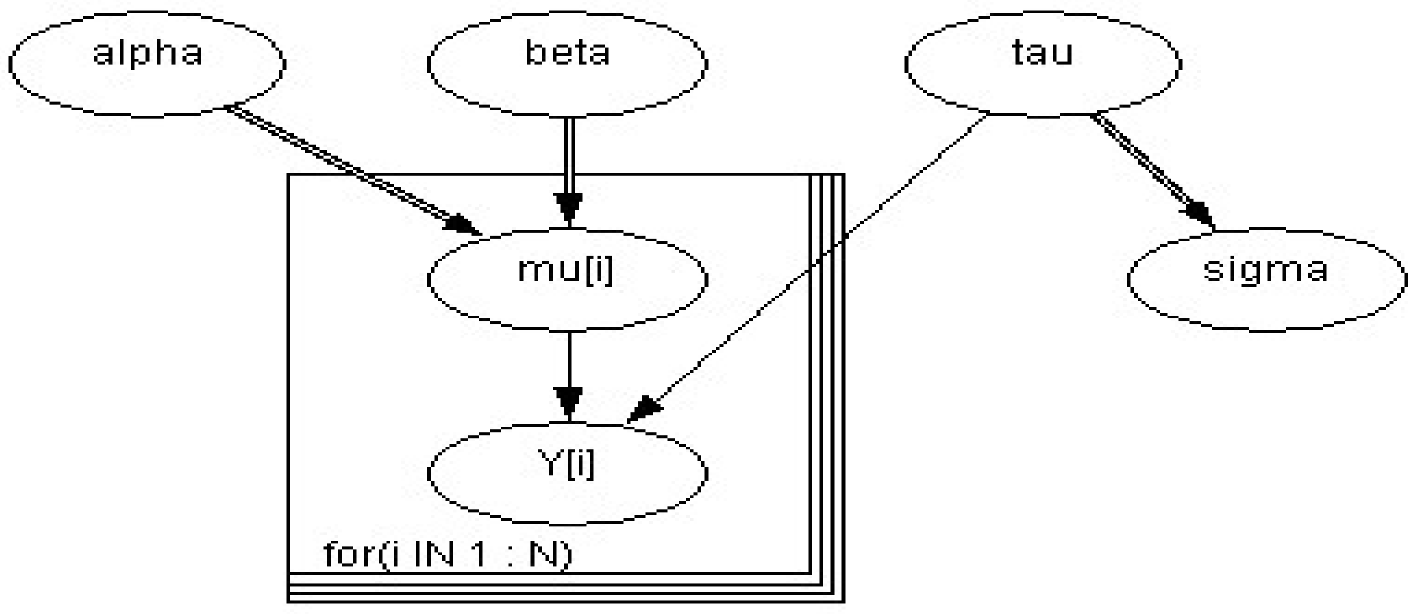
The factorization above implies certain conditional independence restrictions and these can be exploited in the computations. More on this later.

Think of a system of "univariate regressions" (conditional densities to be precise) being put together – this is what we do when writing a BUGS–model.

Notice: The variant of the Ising model given later illustrates that not all models fit into this framework of conditioning.

3 Example: The LINE example from BUGS

Models in BUGS are “DAG–models”.



The plates just symbolize repetitions.

The model in the BUGS language:

```
model
{
        for( i in 1 : N ) {
                Y[i] ~ dnorm(mu[i],tau)
                mu[i] <- alpha + beta * (x[i] - xbar)
        }
        tau ~ dgamma(0.001,0.001) sigma <- 1 / sqrt(tau)
        alpha ~ dnorm(0.0,1.0E-6)
        beta ~ dnorm(0.0,1.0E-6)
}
```

4 Inference

If $p(\theta|y)$ could be found analytically then we would be done – but that is usually not the case (because of the intractable normalizing constant).

Instead we draw samples $\theta^{(1)}, \dots, \theta^{(N)} \sim p(\theta|y)$ and evaluate quantities of interest from these samples. For example

$$\mathbb{E}(\theta|y) \approx \frac{1}{N} \sum_i \theta^{(i)}$$

The tricky part is "how to" draw samples from $p(\theta|y)$ when $p(\theta|y) \propto p(y|\theta)p(\theta)$ is only known up to the normalizing constant.

In some cases simple Monte Carlo methods will do.

Most often, however, we have to use Markov Chain Monte Carlo Methods.

Two specific methods are the Metropolis algorithm and the Metropolis–Hastings algorithm.

A special case of Metropolis–Hastings is the Gibbs sampler.

4.1 Terminology

There are an abundance of Monte Carlo methods available. Here we give some simple examples.

Consider random variable (possibly multidimensional) X with density / probability mass function $p(x)$.

Notice that in most cases $p(x)$ is really a posterior; i.e. $p(x|data)$, but that is not important here.

We shall call $p(x)$ the target distribution (from which we want to sample).

In many (most!) real world applications

- we can not directly draw samples from p.
- p is only known up to a constant of proportionality; that is $p(x) = k(x)/c$ where $k()$ is known and the normalizing constant c is unknown.

We reserve $h(x)$ for a proposal distribution which is a distribution from which we can draw samples.

4.2 Rejection sampling

Suppose we can find a number M such that $p(x) < Mh(x)$ for all y. (Hence $p(x)/(Mh(x)) < 1$). Rejection sampling goes as follows:

1. Draw sample $x \sim h()$.
2. Draw $u \sim U(0,1)$.
3. Accept y if $u < p(x)/(Mh(x))$

The set of accepted values $x^1, \ldots x^N$ generated this way is a sample from p.

Notice:

- It is tricky how to choose a good proposal distribution $h()$. It should have support at least as large as $p()$ and preferably heavier tails than $p()$.
- It is desirable to choose M as small as possible. In practice this is difficult so one tends to take a conservative (large) choice of M whereby only few propsed values are accepted. Thus it is difficult to make rejection sampling efficient.

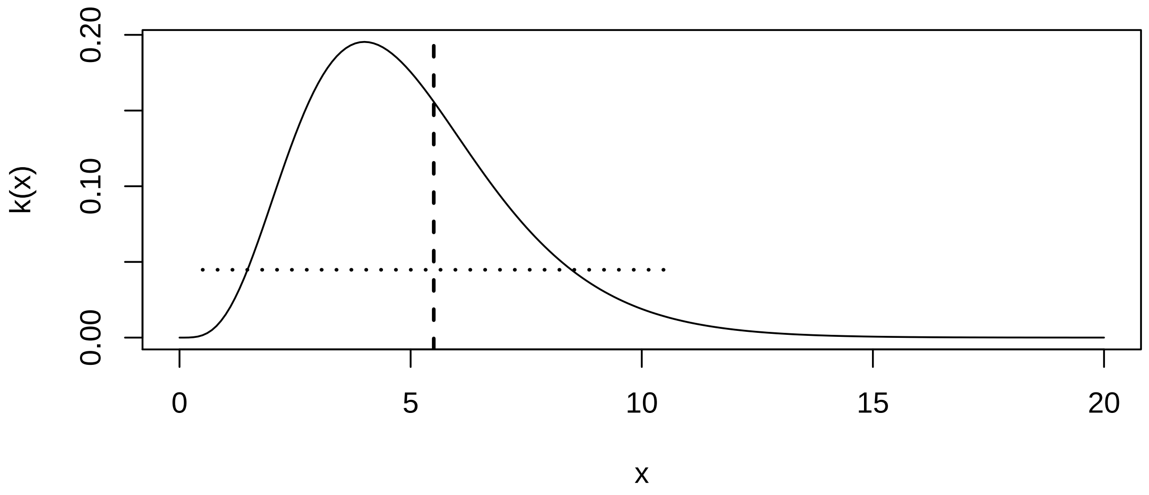
If f is only know up to proportionality ($p(x) = k(x)/c$ and only $k(x)$ is known) then rejection sampling is still an option. Suppose we can find a constant M such that $k(x) < Mh(x)$ for all x. The algorithm is then

1. Draw sample $x \sim h()$.
2. Draw $u \sim U(0, 1)$.
3. Accept x if $u < k(x)/(Mh(x))$

The set of accepted values $x^1, \ldots x^N$ generated this way is a sample from p.

4.3 Slicing

An alternative to rejection sampling is slicing; it works on unnormalized densities as well.



1. Pick a random x–value, say x' (dashed vertical line)
2. Draw $u \sim U(0, k(x'))$ (dotted horizontal line – the slice)
3. Draw x uniformly from the part of the slice under $k(x)$

The set of accepted values $x^1, \ldots x^N$ generated this way is a sample from p.

5 Markov Chain Monte Carlo methods

A major drawback of the rejection algorithm is that it is difficult to suggest a proposal distribution h which leads to an efficient algorithm.

A way around this problem is to let the proposed values depend on the last accepted values: If x' is a “likely” value from p then so is probably also a proposed value x which is close to x'. The price to pay is that we get a Markov chain rather than independent realizations.

Under certain conditions a Markov chain will have a limiting distribution. For a given (DAG-specified) multivariate distribution, the aim is to construct a Markov Chain that has this as limiting distribution.

Notice that the proposal distribution will now be conditional on the last acctepted value and have the form $h(x|x')$.

5.1 The Metropolis algorithm

The Metropolis algorithm goes as follows:

Take a proposal distribution which is symmetrical in its arguments, i.e. $h(x|x') = h(x'|x)$ for all x and x'.

Given an accepted value x^{t-1}:

1. Draw $x \sim h(\cdot|x^{t-1})$.
2. Calculate acceptance probability $\alpha(x|x^{t-1}) = \min(1, \frac{p(x)}{p(x^{t-1})})$
3. Draw $u \sim U(0,1)$
4. If $u < \alpha(x|x^{t-1})$ set $x^t = x$; else set $x^t = x^{t-1}$.

After a burn–in period the samples will be samples from p (but the samples will be correlated).

Notice that that the algorithm also works if p is only known up to proportionality.

The form of the acceptance probability means that if a proposed value x is more likely than the current value x^{t-1} then the proposed value is guaranteed to be accepted whereas if x is less likely than x^{t-1} then x is accepted with a probability being the likelihood ratio between the two values.

5.2 Metropolis–Hastings

The Metropolis–Hastings algorithm is a generalization of the Metropolis algorithm in that there is more flexibility in choosing the proposal distribution. (The proposal needs not be symmetrical).

The algorithm goes as follows:

Given an accepted value x^{t-1}:

1. Draw $x \sim h(\cdot|x^{t-1})$
2. Calculate acceptance probability $\alpha(x|x^{t-1}) = \min(1, \frac{p(x)h(x^{t-1}|x)}{p(x^{t-1})h(x|x^{t-1})})$
3. Draw $u \sim U(0,1)$
4. If $u < \alpha(x|x^{t-1})$ set $x^t = x$; else set $x^t = x^{t-1}$.

After a burn–in period the samples will be samples from p (but the samples will be correlated).

Notice that the algorithm also works if p is only known upto proportionality.

5.3 Single component Metropolis–Hastings

Instead of updating the entire x it is often more convenient and computationally efficient to update x component wise (or more generally in blocks).

Suppose $x = (x_1, x_2, x_3)$. Given an accepted sample $x^{t-1} = (x_1^{t-1}, x_2^{t-1}, x_3^{t-1})$ and that x_1 has also been updated to x_1^t in the current iteration. The task is to update x_2.

1. Draw $x_2 \sim h_2(\cdot|x_1^t, x_2^{t-1}, x_3^{t-1})$
2. Calculate acceptance probability $\alpha(x_2|x_1^t, x_2^{t-1}, x_3^{t-1}) = \min(1, \frac{p(x_2|x_1^t, x_3^{t-1}) h_2(x_2^{t-1}|x_1^t, x_2, x_3^{t-1})}{p(x_2^{t-1}|x_1^t, x_3^{t-1}) h_2(x_2|x_1^t, x_2^{t-1}, x_3^{t-1})})$
3. Draw $u \sim U(0,1)$
4. If $u < \alpha(\cdot|\cdot)$ set $x_2^t = x_2$; else set $x_2^t = x_2^{t-1}$.

5.4 Gibbs sampler

The Gibbs sampler is a special case of single component Metropolis–Hastings.

The proposal distribution $h_2(x_2|x_1^t, x_2^{t-1}, x_3^{t-1})$ for updating x_2 is taken to be

$$p(x_2|x_1^t, x_3^{t-1})$$

- Hence proposed values are always accepted.
- But it requires that we can work out the conditionals $p(x_i|x_{-i})$.

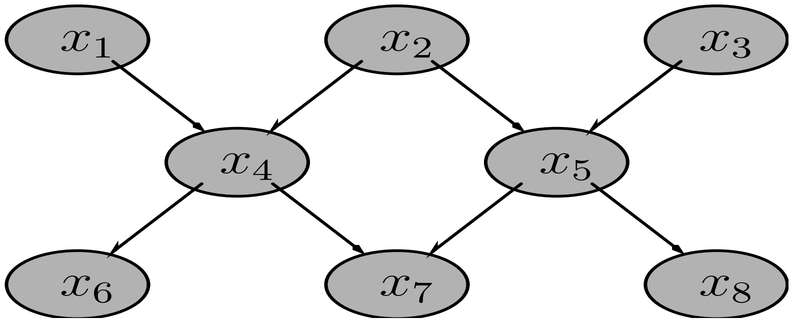
One version of the algorithm is as follows. Suppose a sample $x^t = (x_1^t, x_2^t, x_3^t)$ is available.

1. Sample $x_1^{k+1} \sim p(x_1|x_2^k, x_3^k)$
2. Sample $x_2^{k+1} \sim p(x_2|x_1^{k+1}, x_3^k)$
3. Sample $x_3^{k+1} \sim p(x_3|x_1^{k+1}, x_2^{k+1})$
4. Set $x^{k+1} = (x_1^{k+1}, x_1^{k+2}, x_1^{k+3})$

The sequence $x^1, x^2, \ldots$ then consists of (correlated) samples from $p(x)$.

5.5 DAGs, Gibbs sampling and Metropolis–Hastings

Now return to the DAG; consider updating node x_4.



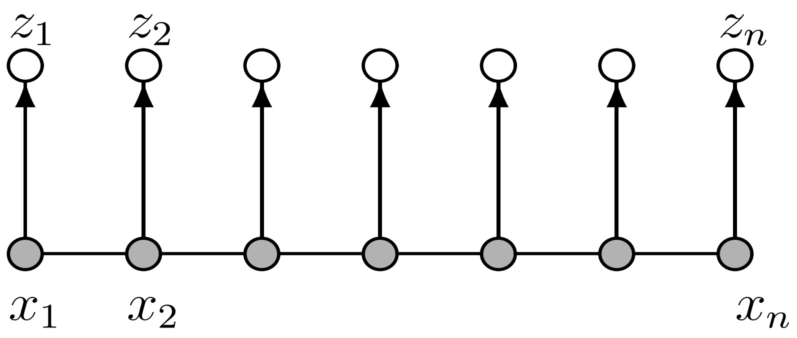
The factorization structure $p = \prod_x p_x(x|parents(x))$ implies certain conditional independencies: x_4 depends “directly on” its parents, its children and its childrens other parents (called the Markov blanket) in the sense that

$$p(x_4|x_1, x_2, x_3, x_4, x_5, x_6, x_7) = p(x_4|x_1, x_2, x_5, x_6, x_7)$$

In Gibbs sampling, the task is to draw one sample from the RHS. In rare cases the RHS simplifies so that this becomes easy. Otherwise a sampling method such as rejection sampling can be used for drawing one such sample.

6 Example: A variant of the Ising model

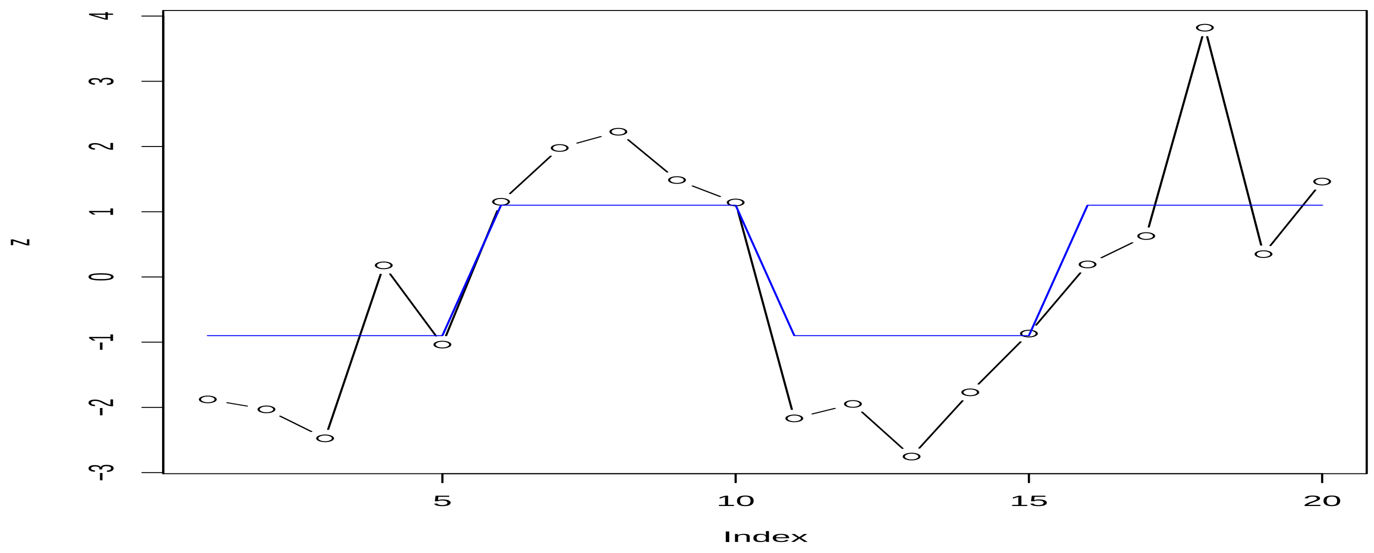
Consider the following model where $x = (x_1, \ldots, x_n)$ is latent and $z = (z_1, \ldots, z_n)$ is observed.



Here x_i interacts only with its neighbours x_{i-1} and x_{i+1}.

The z_is are conditionally independent given x and z_i depends on x only through x_i.

If $x_i \in \{-1, 1\}$ and $z_i | x_i \sim N(x_i, \sigma^2)$ then data could look like:



The assumptions imply that $p(z|x) = \prod_i p(z_i|x_i)$.

As a model for x we take the Ising model:

$$p(x) = \frac{1}{c}\exp(\beta \sum_{i\sim j} 1_{(x_i=x_j)}) = k(x)/c$$

where the sum is over all pairs of xs which are neighbours in the graph.

The normalization constant c is difficult to calculate. Hence $p(z,x) = p(z|x)p(x)$ is only known up to a normalization constant and the same applies for $p(x|z)$.

Notice that when $\beta > 0$ configurations of x where neighbours are identical are given higher probability than configurations where neighbours are different. For now, assume β is known.

6.1 Metropolis algorithm

We use the Metropolis algorithm for finding $p(x|z)$ so the proposal distribution $h(x|x')$ must be symmetrical:

Take e.g. x_{t-1} to be

```
> x.pt <- c(1, 1, 1, -1, -1, -1, -1)
```

A proposal is then to pick a random entry i in x_{t-1} and change the sign of t_i, e.g.

```
> x.ct <- c(1, 1, 1, -1, -1, -1, 1)
```

Starting with all x_is equal to 1 and running Metropolis for 1000 iterations gives the approximate distribution of x

```
                                     config Freq
1  -1+1+1+1+1+1+1+1+1+1+1+1-1-1+1+1+1+1+1+1    2
2  +1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1    9
3  -1-1-1+1-1+1+1+1+1+1+1-1-1-1+1+1+1+1+1+1   10
4  -1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1+1   10
5  -1-1-1+1-1+1+1+1+1+1+1-1-1-1-1+1+1+1+1+1   38
6  -1-1-1+1+1+1+1+1+1+1+1-1-1-1+1+1+1+1+1+1   38
7  -1+1+1+1+1+1+1+1+1+1+1+1+1-1+1+1+1+1+1+1   47
8  -1-1+1+1+1+1+1+1+1+1+1+1-1-1+1+1+1+1+1+1   50
9  -1-1+1+1+1+1+1+1+1+1+1-1-1-1+1+1+1+1+1+1   78
10 -1-1-1-1-1+1+1+1+1+1+1-1-1-1-1+1+1+1+1+1  121
11 -1-1-1-1-1+1+1+1+1+1-1-1-1-1-1+1+1+1+1+1  597
```

If we regard the first 200 iterations as burn-in of the Markov chain and ignore these we get the approximate distribution of x as:

```
                                     config Freq
1 -1-1-1+1-1+1+1+1+1+1+1-1-1-1+1+1+1+1+1+1   10
2 -1-1-1+1+1+1+1+1+1+1+1-1-1-1+1+1+1+1+1+1   34
3 -1-1-1+1-1+1+1+1+1+1+1-1-1-1-1+1+1+1+1+1   38
4 -1-1-1-1-1+1+1+1+1+1+1-1-1-1-1+1+1+1+1+1  121
5 -1-1-1-1-1+1+1+1+1+1-1-1-1-1-1+1+1+1+1+1  597
```

7 Odds and ends

7.1 Sampling importance resampling (SIR)*

Even when M is not readily available, we may generated approximate samples from p as follows.

1. Draw samples $x^1, \dots x^N \sim h(x)$.
2. Calculated importance weights $w_i = p(x^i)/h(x^i)$.
3. Normalize the weights as $q_i = w_i / \sum_j w_j$.
4. Resample from $\{x^1, \dots x^N\}$ where y^i is drawn with probability q_i.

These last samples are approximately samples from p.

This scheme works also if p is only known up to proportionality.

Notice that those samples from h which "fits best to p" are those most likely to appear in the resample. But it is also worth noticing that if h is a poor approximation to p then then the "best samples from h" are not necessarily good samples in the sense of resembling p.

7.2 Random–walk Metropolis*

This is a special case of the Metropolis algorithm and arises if $h(x|x') = h(|x - x'|)$.

7.3 The independence sampler*

The independence sampler is a special case of Metropolis Hastings where the proposal $h(x|x') = h(x)$ does not depend on x'.

The acceptance probability becomes

$$\alpha(x|x^{t-1}) = \min(1, \frac{p(x)h(x^{t-1})}{p(x^{t-1})h(x)})$$

For this sampler to work well, h should be a good approximation to p.