CS294 Markov Chain Monte Carlo: Foundations & Applications **Fall 2009**

Lecture 13: Oct 20

Lecturer: Alistair Sinclair *Scribes: Alistair Sinclair*

Last time we saw an MCMC algorithm that samples matchings M in an undirected graph $G = (V, E)$ from the Gibbs distribution $\pi_\lambda(M) = \frac{1}{Z(\lambda)}\lambda^{|M|}$ with running time poly(n, λ), where $n = |V|$. Here the normalizing factor $Z(\lambda) = \sum_M \lambda^{|M|} = \sum_k m_k \lambda^k$, where m_k is the number of k-matchings in G, is known in the statistical physics context as the *partition function*, and in the combinatorial context as the *matching polynomial*. In this lecture we look at some applications and extensions of this Markov chain on matchings, taken from [JS89].

13.1 Extensions and Applications

13.1.1 Estimating the Partition Function Z(λ)

Computing $Z(\lambda)$ exactly for any fixed value of $\lambda > 0$ is #P-complete. So the best we can hope for is an approximation algorithm.

First, we express $Z(\lambda)$ as a telescoping product as follows:

$$Z(\lambda) = \frac{Z(\lambda_r)}{Z(\lambda_{r-1})} \times \frac{Z(\lambda_{r-1})}{Z(\lambda_{r-2})} \ldots \times \frac{Z(\lambda_1)}{Z(\lambda_0)} \times Z(\lambda_0), \text{ where } 0 = \lambda_0 < \lambda_1 < \lambda_2 < \ldots < \lambda_r = \lambda.$$

Notice that $Z(\lambda_0) = Z(0) = 1$ because there is exactly one 0-matching (the empty matching) in any G. The sequence λ_i is chosen to increase slowly so that each factor in the product is bounded by a constant, allowing it to be estimated efficiently by random sampling.

Accordingly, define the sequence as: $\lambda_1 = \frac{1}{|E|}, \lambda_i = (1+\frac{1}{n})\lambda_{i-1}$ for $i = 2, \ldots, r-1$, and $\lambda_r = \lambda \le (1+\frac{1}{n})\lambda_{r-1}$. Notice that this gives $r = O(n \log \lambda)$, so there are not many factors in the product. Also, we can easily verify the following upper bound for each factor:

$$\frac{Z(\lambda_i)}{Z(\lambda_{i-1})} = \frac{\sum_k m_k \lambda_i^k}{\sum_k m_k \lambda_{i-1}^k} \le \left(1 + \frac{1}{n}\right)^n \le \mathrm{e}.$$

The final observation is that each such factor can be expressed as the expectation of an appropriate random variable defined over the Gibbs distribution on matchings with parameter λ_{i-1} This implies that we can estimate each factor by random sampling from the Gibbs distribution π_{λ_i}.

Claim 13.1 $\frac{Z(\lambda_i)}{Z(\lambda_{i-1})} = E_{\pi_{\lambda_{i-1}}}\left[\left(\frac{\lambda_i}{\lambda_{i-1}}\right)^{|M|}\right]$

Proof:

$$\begin{aligned}
E_{\pi_{\lambda_{i-1}}}\left[\left(\frac{\lambda_i}{\lambda_{i-1}}\right)^{|M|}\right] &= \frac{1}{Z(\lambda_{i-1})}\sum_k m_k \lambda_{i-1}^k \left(\frac{\lambda_i}{\lambda_{i-1}}\right)^k \\
&= \frac{1}{Z(\lambda_{i-1})}\sum_k m_k \lambda_i^k \\
&= \frac{Z(\lambda_i)}{Z(\lambda_{i-1})}.
\end{aligned}$$

■

Our algorithm will estimate $Z(\lambda)$ by estimating each $Z(\lambda_i)/Z(\lambda_{i-1})$ in succession and taking the product as above. Each of these factors is estimated by random sampling from π_{λ_i}. Since, as we saw above, the expectation of each such r.v. is bounded by a constant, the number of samples required for each factor in order to ensure an overall estimate within a factor of $(1 \pm \epsilon)$ is about $O(n^2\epsilon^{-2})$. (See the analogous calculation in a future lecture for graph colorings.) Also, by our mixing time analysis, the time to obtain each sample is bounded by poly(n, λ).

The bottom line is a *fully-polynomial randomized approximation scheme* for $Z(\lambda)$, that is, we get an estimate of $Z(\lambda)$ within ratio $(1 \pm \epsilon)$ with high probability in total time poly$(n, \lambda, \epsilon^{-1})$.

13.1.2 Estimating the Coefficients m_k

The coefficient m_k represents the number of k-matchings in the graph. When $k = |V|/2$, m_k is the number of perfect matchings. If the graph G is bipartite with $|V|$ vertices on each side, counting the number of perfect matchings is equivalent to computing the *permanent* of the adjacency matrix A_G of G. (This is an easy **exercise**.)

Definition 13.2 *The permanent of an $n \times n$ matrix A is defined as*

$$\text{per}(A) = \sum_\sigma \prod_{i=1}^n A(i, \sigma(i)),$$

where the sum is over all permutations σ of $\{1, \ldots, n\}$.

Note the similarity with the determinant:

$$\det(A) = \sum_\sigma (-1)^{sign(\sigma)} \prod_{i=1}^n A(i, \sigma(i)).$$

However, while we have several polynomial time algorithms (such as Gaussian elimination) for computing $\det(A)$, it is known that evaluating per(A) is #P-complete. (This is a celebrated result of Valiant [Val79a].)

We return now to estimating m_k. If we could compute the partition function $Z(\lambda)$ *exactly*, we could also get all the m_k exactly by computing $Z(\lambda)$ at $n + 1$ points λ and using interpolation. (Note that Z is a polynomial of degree at most n.) However, polynomial interpolation is not robust, so if we only know the value of $Z(\lambda)$ approximately this could result in large errors in its coefficients. Instead we use a different approach. We need the following two claims before describing the algorithm.

Claim 13.3 *The sequence $\{m_k\}$ is* log-concave*: i.e., $m_{k-1}m_{k+1} \le m_k^2$ for all k.*

This is left as an **exercise**. [Hint: set up an injective mapping that takes a pair of matchings, one with $k-1$ edges and the other with $k+1$ edges, to a pair of k-matchings. This uses similar ideas to our encoding analysis for the mixing time in the last lecture.]

Claim 13.4 *If $\lambda = \frac{m_{k-1}}{m_k}$, then the ratio $m_{k'}\lambda^{k'}$ is maximized at $k' = k$ and $k' = k-1$.*

Proof: First notice that $m_k\lambda^k = (m_k\lambda)\lambda^{k-1} = m_{k-1}\lambda^{k-1}$. Next, since the log function is monotonic, it suffices to check that $\log(m_{k'}\lambda^{k'}) = \log m_{k'} + k'\log\lambda$ is maximized at $k' = k$. Since the sequence $\{\log m_{k'}\}$ is concave (by the previous Claim), it suffices to show that $m_k\lambda^k \geq m_{k+1}\lambda^{k+1}$ and that $m_{k-1}\lambda^{k-1} \geq m_{k-2}\lambda^{k-2}$. To see the first of these, note that by log-concavity

$$\frac{m_{k+1}\lambda^{k+1}}{m_k\lambda^k} = \lambda\frac{m_{k+1}}{m_k} = \frac{m_{k-1}m_{k+1}}{m_k^2} \leq 1.$$

The argument for the second inequality is entirely similar. ∎

This suggests the following algorithm to successively estimate m_k. Trivially $m_0 = 1$, so we can estimate m_k by successively estimating each of the ratios $m_{k'}/m_{k'-1}$ for $2 \leq k' \leq k$ and multiplying them together. To estimate the ratio $m_{k'}/m_{k'-1}$, the idea is to raise λ until $(k'-1)$- and k'-matchings appear frequently enough in the stationary distribution π_λ to estimate their ratio reliably:

- Gradually increase λ and sample from the Gibbs distribution π_λ until we see "lots of" $(k'-1)$- and k'-matchings in stationary distribution. "Lots of " means that their probability is at least about $\frac{c}{n}$ for some constant c.
- For this λ, use samples from the MC to estimate $m_{k'}/m_{k'-1}$. The estimator is the ratio of the number of k'-matchings to the number of $(k'-1)$-matchings in the sample, multiplied by the factor $1/\lambda$.
- To obtain the estimate of m_k, multiply the estimates of $m_{k'}/m_{k'-1}$ obtained as above for $2 \leq k' \leq k$.

Claim 13.4 confirms that, in each stage of the above algorithm, λ won't need to be larger than $m_{k'-1}/m_{k'}$, which by log-concavity is at most m_k/m_{k-1} for all $k' \leq k$. By our Markov chain analysis from the last lecture, the time to obtain a sample is polynomial in n and λ, and by similar standard statistical arguments as in the previous section, the total number of samples required for a $(1\pm\epsilon)$ estimate of m_k is polynomial in n and ϵ^{-1}. Hence the entire algorithm takes time poly$(n, m_{k-1}/m_k, \epsilon^{-1})$ to get a $(1\pm\epsilon)$ estimate of m_k/m_{k-1} with high probability. Thus the algorithm works well as long as m_{k-1}/m_k is not too large, i.e., smaller than some fixed polynomial poly(n).

Exercises: (i) In a graph with $2n$ vertices (so that a maximum matching has size n), show how to use the above algorithm to estimate $m_{(1-\epsilon)n}$ in time $n^{O(1/\epsilon)}$.
(ii) Show how to use the above algorithm to find a matching of size at least $(1-\epsilon)k^*$ in any given graph, where k^* is the (unknown) size of a maximum matching in the graph. Your algorithm should run in time $n^{O(1/\epsilon)}$.

Let us now focus on the important problem of counting *perfect* matchings, which as we have noted is equivalent to computing the permanent of a 0-1 matrix. We will assume that our graph has $|V| = 2n$ vertices, so the size of a perfect matching is n. Our goal is thus to compute m_n. The above algorithm will run in polynomial time provided the ratio m_{n-1}/m_n is polynomially bounded. This property happens to hold for many interesting classes of graphs, including, for example, dense graphs (all vertex degrees are at least $n/2$), random graphs (in the $G_{n,1/2}$ model, with high probability), and regular lattices (e.g., finite square regions of the grid Z^2, which are important in physical applications such as the dimer model). However, unfortunately it is not too hard to construct examples of graphs where this ratio grows exponentially with n.

The figure below shows such an example. The graph consists of ℓ squares connected in a line (so the number of vertices is $2n = 4\ell + 2$). There is a single perfect matching, but the number of "near-perfect matchings" (i.e., those of size $n-1$) is at least $2^\ell = 2^{(n-1)/2}$ (corresponding to leaving the two endpoints unmatched and having two choices on each square). Thus $m_{n-1}/m_n \geq 2^{(n-1)/2}$.

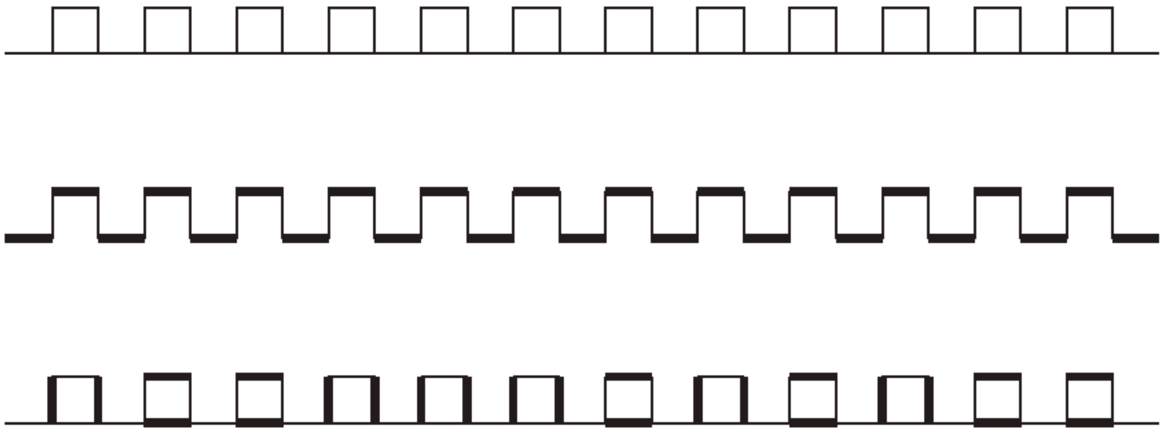

Figure 13.1: The first row shows a graph; the second row shows its unique perfect matching; the third row shows one of many near-perfect matchings

13.2 Perfect Matchings in an Arbitrary Graph / Permanent of Arbitrary Positive Matrix

How can we handle graphs in which the ratio m_{n-1}/m_n of near-perfect matchings to perfect matchings is very large? This was done in [JSV04] by an extension of the above MCMC approach for all matchings. The key new idea is to reweight the matchings in such a way that the perfect matchings have large enough weight in the stationary distribution; the problem is that the required weights are very hard to compute, but we can get the MC itself to succcessively *learn* the correct weights.

Our starting point is a slightly different MC for sampling k-matchings to the one above (whose states were all the matchings, of all sizes). This MC has as states only the k- and $(k-1)$-matchings, and its stationary distribution is uniform. The transitions (edge additions, deletions and exchanges) are the same as for our previous chain. By an analysis very similar to the one we used for the all-matchings MC, it can be shown that the mixing time for this MC is poly$(n, \frac{m_{k-1}}{m_k})$. Since we are interested in perfect matchings, we will focus on the chain whose states are perfect and near-perfect matchings (i.e., $k = n$).

We partition the set of near-perfect matchings into sets $\{N(u,v)\}$, where $N(u,v)$ denotes the set of near-perfect matchings with "holes" (unmatched vertices) at u and v. Let P be the set of perfect matchings. A typical distribution of $\{|N(u,v)|\}$ and $|P|$ (with all matchings equally weighted) is sketched on the left side of Figure 13.2. Note that in a "bad" graph at least one of the $|N(u,v)|$ is exponentially larger than $|P|$.

Now assign to each matching M in the graph a weight $w(M)$ defined as:

$$w(M) = \begin{cases} w(u,v) := \frac{|P|}{|N(u,v)|} & \text{if } M \in N(u,v); \\ 1 & \text{if } M \in P; \\ 0 & \text{otherwise.} \end{cases}$$

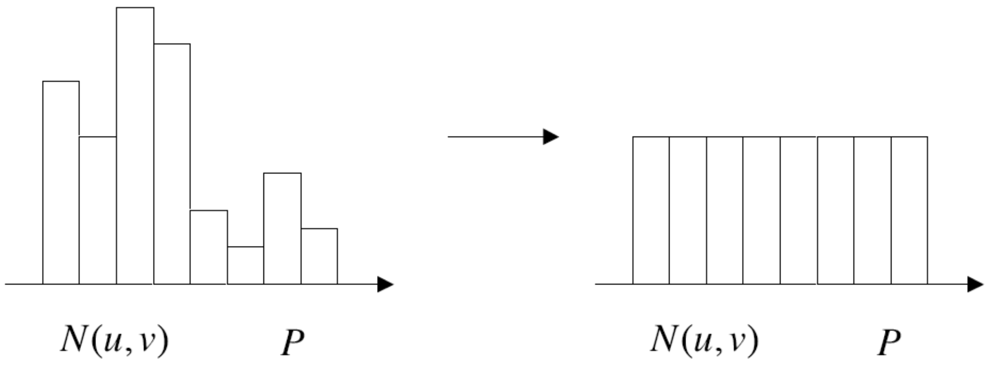

Figure 13.2: The histogram of $\{|N(u,v)|\}$ and $|P|$ before and after reweighting.

These weights are chosen precisely so that the weight of each of the sets $N(u,v)$ is the same, and equal to that of the perfect matchings P. In other words, we have reweighted the distribution so that it looks like the right-hand side of Figure 13.2. We can easily modify our Markov chain so that it has this distribution as its stationary distribution (i.e., $\pi(M) \propto w(M)$) by using the Metropolis rule (i.e., transitions are accepted with probability proportional to the ratio of the weights of the two matchings). This Markov chain has two nice properties: (i) it allocates total weight at least $\frac{\text{const}}{n^2}$ to the set of perfect matchings P (because there are at most n^2 different possible hole patterns (u,v)); and (ii) the mixing time is poly(n) (this follows by a flow argument similar to the one we used for the all-matchings chain but a bit more technical; the weights on the matchings have the effect of cancelling the factor m_{n-1}/m_n from the mixing time).

So we have a rapidly mixing Markov chain that samples perfect matchings u.a.r. with good probability—and it seems that we are done! But there is a catch. In order to implement the above algorithm we need to know the weights $w(u,v)$ (for the Metropolis rule). But these are defined as ratios of the form $\frac{|P|}{|N(u,v)|}$, which involve quantities like $|P|$ which is what we are trying to compute in the first place! So how can we get hold of the weights?

The final trick here is to introduce edge activities λ_e, so that the weights also depend on these. We start off with trivial activities $\lambda_e = 1$ for *all possible* edges (even those that are not in G). For these trivial activities, the weights $w(u,v)$ are easy to compute. Then we gradually reduce the activities of the non-edges until they become negligible. As we do this, we are able to *learn* the weights $w(u,v)$ for the new activities using observations from the MC itself!

Here is a summary of the procedure:

- Introduce edge activities λ_e for all $e \in |V| \times |V|$, and define:

$$\lambda(M) := \prod_{e \in M} \lambda_e; \quad \lambda(P) := \sum_{M \in P} \lambda(M); \quad \lambda(N(u,v)) := \sum_{M \in N(u,v)} \lambda(M).$$

  Here $\lambda(M)$ is the activity of a matching, and $\lambda(P)$, $\lambda(N(u,v))$ are just weighted versions of the cardinalities $|P|$ and $|N(u,v)|$ from before.

- Redefine the hole weights as $w_\lambda(u,v) := \frac{\lambda(P)}{\lambda(N(u,v))}$. This introduction of activities λ has essentially no effect on the mixing time analysis, so the Markov chain still mixes in polynomial time. The stationary distribution becomes $\pi_\lambda(M) \propto \lambda(M) w_\lambda(u,v)$.

- Start with $\lambda_e = 1$ for all edges e (including "non-edges" $e \notin E(G)$). In other words, we start out with the complete bipartite graph $K_{n,n}$. For this graph, computing the hole weights $w_\lambda(u, v)$ is trivial.
- Gradually reduce the activities λ_e of non-edges $e \notin E(G)$ until $\lambda_e \ll 1/n!$; at this point the stationary weight of any matching containing a non-edge is negligible, so we are effectively just sampling perfect and near-perfect matchings in the graph G. Since the activities of all the edges in G are 1, the stationary distribution is essentially the same as in the right-hand side of Figure 13.2. We reduce the activities in stages: at each stage we reduce λ_e for just one of the non-edges e by a factor of 2. The number of stages we need is thus about $O(n^3 \log n)$.
- Assume inductively that we have the correct hole weights for the current set of activities $\{\lambda_e\}$. Now we reduce one of the λ_e, so the hole weights are no longer correct. However, it is not hard to see that reducing a single λ_e by a factor of 2 can affect each of the hole weights by at most a factor of 2 as well. So we can continue to use our old hole weights with the new activities. The constant factor error in hole weights just means that the stationary distribution will be slightly biased (by at most a constant factor), and that the mixing time may increase (again by at most a constant factor). So we still have a rapidly mixing Markov chain for the new activities. Now the crucial point is that, by taking samples from the stationary distribution of this Markov chain, we can easily estimate the *correct* hole weights for the new activities. Hence we can get very accurate estimates of these, and we are back in the situation we started in but with the new activities.

References

[1] M. Jerrum and A. Sinclair, "Approximating the permanent." *SIAM Journal on Computing* **18** (1989), pp. 1149–1178.

[2] M. Jerrum, A. Sinclair and E. Vigoda, "A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries." *Journal of the ACM* **51** (2004), pp. 671–697.

[V79a] L.G. Valiant, "The complexity of computing the permanent," *Theoretical Computer Science* **8**(1979), pp. 189–201.