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

Lecture 12: October 15

Lecturer: Alistair Sinclair *Scribes: Alistair Sinclair*

12.1 Flow encodings

Recall the generic calculation from the end of last lecture: in a typical Markov chain of the type that occurs in our applications, if the stationary distribution is uniform and if we send the $x \to y$ flow along a single path γ_{xy}, then in order for this flow to be good it must satisfy

$$|\mathrm{paths}(e)| \leq \mathrm{poly}(n)|\Omega| \qquad \forall e, \tag{12.1}$$

where paths(e) is the set of paths that go through edge e. If we allow multiple paths from x to y then the same calculation applies but in an average sense, and if the stationary distribution is non-uniform then again the same applies but with everything suitably weighted.

Equation (12.1) indicates that we must compare $|\mathrm{paths}(e)|$ with the size of the state space $N = |\Omega|$. However, since computing N is often our goal, this seems difficult. The following machinery is designed to get around this problem. The key idea is to set up an *injective mapping* from paths(e) to Ω which allows us to compare their sizes implicitly. This technology is employed in almost all non-trivial applications of multicommodity flows in the analysis of mixing times.

Definition 12.1 *An* encoding *for a flow f (that uses only single paths γ_{xy} for each x, y) is a set of functions $\eta_e : \mathrm{paths}(e) \to \Omega$ (one for each edge e) such that*

1. *η_e is injective*
2. *$\pi(x)\pi(y) \leq \beta\pi(z)\pi(\eta_e(x,y))\ \forall (x,y) \in \mathrm{paths}(e)$, where $e = (z, z')$.*

Remark: Property 1 says that η_e is an injection, as motivated earlier. Property 2 says that η_e is in addition weight-preserving up to a factor β (which is assumed to be not too large: constant or polynomially bounded). Note that property 2 is automatically satisfied with $\beta = 1$ when π is uniform. In some applications, we may usefully weaken property 1 slightly as follows: we may not have a perfect injection, but may require a small amount of "extra information" to invert η_e. As should be clear from the proof of the following claim, this will just insert a modest additional factor into the bound on the mixing time.

Claim 12.2 *If there exists an encoding for f as above, then $\rho(f) \leq \beta \max_{P(z,z')>0} \frac{1}{P(z,z')}$.*

Proof: Let $e = (z, z')$ be an arbitrary edge. Then

$$f(e) = \sum_{(x,y)\in\mathrm{paths}(e)} \pi(x)\pi(y) \leq \beta \sum_{(x,y)\in\mathrm{paths}(e)} \pi(z)\pi(\eta_e(x,y)) \leq \beta\pi(z).$$

In the first inequality here we have used property 2, and in the second we have used property 1.

Finally, $C(e) = \pi(z)P(z,z')$, so $f(e)/C(e) \le \beta/P(z,z')$. ■

We now give a simple example of an analysis using flow encodings, followed by a more serious example.

12.2 Example: Cube $\{0,1\}^n$

Previously, in analyzing the random walk on the cube by flows, we spread flow evenly and used the symmetry of the cube to bound $\rho(f)$. In more sophisticated applications, we will not have this symmetry and not know $N = 2^n$. Flow encodings let us proceed without appealing to these properties of the cube.

Recall that the capacity of an edge e is $C(e) = 1/(2nN)$, and the demand between two vertices x, y is $D(x,y) = 1/N^2$. Now consider a flow that sends all the $x \to y$ flow along the "left-right bit-fixing path" γ_{xy}. That is, correct each bit of x sequentially, left-to-right, until arriving at y. Then $\ell(f) = n$, clearly.

We now bound the cost of this flow using the encoding technique. Consider an arbitrary edge $e = (z,z')$, where z and z' differ in bit position i. Consider any pair $(x,y) \in \text{paths}(e)$. What do we know about x and y? Notice that y agrees with z in the first $i-1$ bits (which have already been corrected), and x agrees with z in the last $n-i$ bits. We therefore define $\eta_e : \text{paths}(e) \to \Omega$ by $\eta_e(x,y) = x_1x_2\ldots x_iy_{i+1}y_{i+2}\ldots y_n$. I.e., $\eta_e(x,y)$ is the $0,1$-string that agrees with x on the first i bits and with y on the rest.

Now it is easy to see that we can uniquely recover (x,y) from $\eta_e(x,y)$ and e (this is why we constructed η_e this way!), so η_e is an injection. Since the stationary distribution is uniform, it is trivially weight-preserving. Hence η_e is a valid encoding. By the above Claim therefore,

$$\rho(f) \le \max_{z,z'} \frac{1}{P(z,z')} = 2n \ .$$

Up to a constant, this is the same bound we obtained for the cube by spreading flow uniformly and appealing to symmetry. In particular, we again get $\tau_{\text{mix}} \le O(n^3)$. However, the key point about this alternative flow and its analysis is that we never used the fact that $N = 2^n$, nor any of the cube's symmetry. In the next section, we'll see a much harder example in which this feature is crucial.

12.3 Matchings in a graph

We now use encodings to analyze a more complicated Markov chain that samples matchings in a graph, due to [JS89].

Recall that a *matching* in G is a set of edges of G such that no two edges share a vertex. Given an undirected graph $G = (V,E)$ and a parameter $\lambda \ge 1$,[1] we wish to sample from the set Ω of all matchings in G according to the *Gibbs distribution*,

$$\pi(M) = \frac{1}{Z}\lambda^{|M|},$$

where $|M|$ is the number of edges in the matching M and $Z = Z(\lambda)$ is the normalizing factor (partition function). If m_k is the number of k-matchings of G (matchings with k edges), then $Z(\lambda) = \sum_k m_k\lambda^k$, the matching polynomial of G. This problem is motivated both by its combinatorial significance and because it corresponds to the so-called *monomer-dimer* model of statistical physics: in this model, vertices connected by

[1]Actually the algorithm and its analysis are essentially the same for $\lambda < 1$, but we consider only the more important case $\lambda \ge 1$ for definiteness.

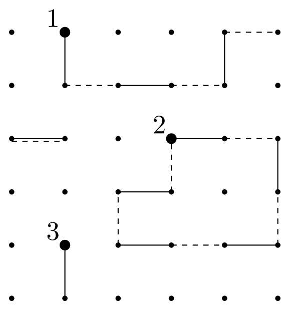

Figure 12.1: The matchings x (solid lines) and y (dotted lines).

an edge in the matching correspond to diatomic molecules (dimers), and unmatched vertices to monatomic molecules (monomers). We note also that computing the partition function $Z(\lambda)$ is #P-complete for any fixed $\lambda > 0$.

Markov chain: We define a Markov chain on the space of matchings using three kinds of transitions: edge addition, edge deletion, and edge exchange (deleting an edge and adding an edge sharing one vertex with the deleted edge). We make the Markov chain lazy and use the Metropolis rule to make the stationary distribution match the Gibbs distribution, as follows. At a matching $M \in \Omega$,

- (Laziness) With probability 1/2, stay at M.
- Otherwise, choose an edge $e = (u, v) \in E$ u.a.r.
- (Edge addition) If both u and v are unmatched in M, then go to $M + e$.
- (Edge deletion) If $e \in M$, then go to $M - e$ with probability $1/\lambda$ (else stay at M).
- (Edge exchange) If exactly one of u and v is matched in M, let e' be the unique edge in M containing u or v, and go to $M + e - e'$.
- If both u and v are matched, then do nothing.

This Markov Chain follows the Metropolis rule. Edge addition and edge exchange either increase or do not change the weight of the matching, so we always perform these moves. However, an edge deletion decreases the weight by a factor of λ, so we only accept an edge deletion with probability $1/\lambda$.

Flow: To define a flow f, we pick a path γ_{xy} for every pair of matchings x and y and send all of the flow along it. Imagine that we color x red and y blue, then superimpose the two matchings to form $x + y$. Because x and y are matchings, the connected components of $x + y$ consist of (closed) cycles of alternating red and blue edges, (open) paths of alternating red and blue edges, and edges that are both red and blue (or equivalently, trivial alternating cycles of length two).

Now fix (for the sake of analysis only) an arbitrary total ordering on the set of *all* open paths and even-length cycles (of length at least four) in G. Designate one vertex of each such cycle and path as its "start vertex"; the start vertex must be an endpoint in the case of a path. This ordering induces an ordering on those paths and cycles that actually appear in $x+y$. Figure 12.1 shows an example; the start vertices in each component are marked with a large circle and the induced ordering is shown.

To define the flow from x to y, we process the paths and cycles in the given order. To process a path, let its consecutive edges be $e_1, e_2, \ldots, e_r$, where e_1 is the edge containing the start vertex, and apply the following transitions to x:

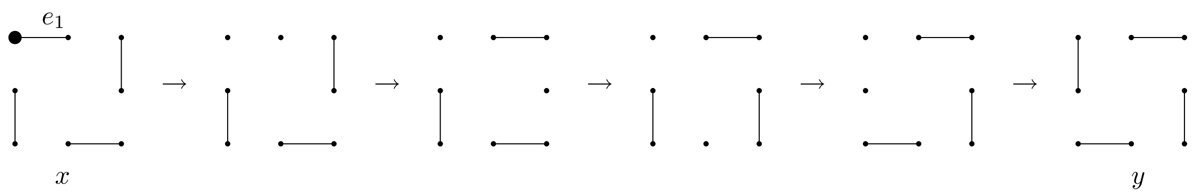

Figure 12.2: Unwinding a single cycle along the path from x to y.

- If e_1 is red, then remove e_1, exchange e_3 for e_2, exchange e_5 for e_4, and so on. If e_r is blue, we have the additional move of adding e_r at the end.
- If e_1 is blue, then exchange e_2 for e_1, exchange e_4 for e_3, and so on. If e_r is blue, we have the additional move of adding e_r at the end.

For cycles, the processing is similar. Suppose that the edges of the cycle are $e_1, \ldots, e_{2r}$, where e_1 is the red edge adjacent to the start vertex. Process the cycle by first removing e_1, then exchanging e_3 for e_2, exchanging e_5 for e_4, …, exchanging e_{2r-1} for e_{2r-2}, and finally adding e_{2r}.

By processing the components of $x+y$ in succession, we have constructed a path from x to y; let γ_{xy} be this path.

Encoding: We will define the encoding η_t for a transition $t = (z, z')$ corresponding to an edge exchange. For the other types of transition t, η_t is defined similarly.

Suppose transition t involves exchanging e' for e. Now let x and y be matchings for which the path γ_{xy} traverses t, i.e., $(x,y) \in \text{paths}(t)$. Consider the multiset $x+y$. If z and z' are part of a cycle of $x+y$, then let e_1 be the first edge removed in processing $x+y$, and define

$$\eta_t(x,y) = (x+y) \setminus (z \cup z' \cup \{e_1\}).$$

Otherwise, define

$$\eta_t(x,y) = (x+y) \setminus (z \cup z').$$

It is not hard to check [**exercise!**] that $\eta_t(x,y)$ is always a matching, so η_t is well-defined. Now this definition may seem mysterious, but $\eta_t(x,y)$ is really the disjoint union of $x \cap y$ (i.e., all edges of $x+y$ that are both red and blue) and $(x \oplus y) \setminus (z \cup z')$ (with e_1 also deleted if we are processing a cycle). The latter term is just all the edges in the paths and cycles of $x+y$ that are missing from $z \cup z'$. It will be convienient for us to write this as

$$\eta_t(x,y) = x \oplus y \oplus (z \cup z') \setminus \{e_1\}. \tag{12.2}$$

As an example, let x and y be as in Figure 12.2 and let t be the middle (third) transition in the example. Then the encoding $\eta_t(x,y)$ is as shown in Figure 12.3.

We need to check that η_t is indeed an encoding. To verify the injectivity property, we need to be able to recover (x,y) uniquely from $\eta_t(x,y)$. Using (12.2), we see that $x \oplus y = (\eta_t(x,y) \cup \{e_1\}) \oplus (z \cup z')$, so we can recover $x \oplus y$. (There is a detail here: because of the absence of edge e_1 from both $z \cup z'$ and $\eta_t(x,y)$, we can't tell the difference between a cycle and a path that differ up to addition of e_1. But we can overcome this with a trick: we can use the sense of unwinding of the path/cycle to distinguish the two possibilities.) Now we can partition $x \oplus y$ into x and y by using the ordering of the paths. That is, from the transition t we can identify which path/cycle is currently being processed. Then for every path that preceeds the current one, we know that the path agrees with y in z and with x in $\eta_t(x,y)$. For the paths following the current one, the parity is reversed. Finally, for the current path we know that z agrees with y up to the point of the

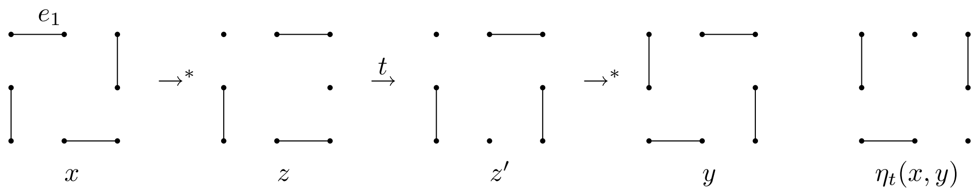

Figure 12.3: The transition t and its encoding $\eta_t(x, y)$.

current transition, and with x beyond it. Finally, we can easily identify the edges that belong to both x and y as they are just $z \cap z' \cap \eta_t(x, y)$. Thus we see that η_t is indeed an injection.

Notice that for a transition $t = (z, z')$ on a path from x to y, we have that the multiset $z \cup \eta_t(x, y)$ has at most two fewer edges than the multiset $x \cup y$ (arising from the possible edge deletion that starts the processing of a path, and from the edge missing from the current transition). Therefore, $\pi(x)\pi(y) \le \lambda^2 \pi(z)\pi(\eta_t(x, y))$, so η satisfies property (ii) in the definition of encoding, with $\beta = \lambda^2$.

Analysis of τ_{mix}: For any transition $t = (z, z')$, $P(z, z') \ge \frac{1}{2\lambda|E|}$ (attained for edge deletions t). Thus by the above encoding and Claim 12.2,

$$\rho(f) \le \lambda^2 \cdot \left(\frac{1}{2\lambda|E|}\right)^{-1} = O(\lambda^3|E|).$$

Now the length of γ_{xy} is at most $|x| + |y|$ because every edge of $x \cup y$ is processed at most once. Since x and y are matchings, $|x|$ and $|y|$ are at most $|V|/2$. Hence the length of γ_{xy} is at most $|V|$, and $\ell(f) \le |V|$. By Theorem 11.2 of Lecture 11, the Poincaré constant α of P satisfies

$$\alpha \ge \frac{1}{\rho(f)\ell(f)} \ge \Omega\left(\frac{1}{\lambda^3|V||E|}\right).$$

Then Corollary 11.3 of Lecture 11 implies that

$$\tau_x(\epsilon) \le O\left((\log \pi(x)^{-1} + \log \epsilon^{-1}) \cdot \frac{1}{\alpha}\right).$$

From this we obtain

$$\tau_x(\epsilon) \le O\left(\lambda^3|E||V|(\log \pi(x)^{-1} + \log \epsilon^{-1})\right).$$

Note that Ω is a subset of the $2^{|E|}$ subgraphs of G, so $|\Omega| \le 2^{|E|}$. Also, there exists a polynomial time algorithm to find a matching x of G with the maximum number of edges. This x has maximal weight in the Gibbs distribution, so

$$\pi(x) \ge \frac{1}{|\Omega|} \ge \frac{1}{2^{|E|}}.$$

Therefore, the mixing time starting from a maximum matching x is bounded by

$$O\left(\lambda^3|E|^2|V|\right),$$

which is polynomial in λ and in the size of the graph. [As a side note, we can reduce the upper bound on the mixing time to $O\left(\lambda^2|E|^2|V|\right)$ by taking a little more care in the above encoding analysis.]

In the next lecture, we will look at some applications and extensions of this Markov chain on matchings.

References

[JS89] M. Jerrum and A. Sinclair, “Approximating the Permanent,” *SIAM Journal on Computing* **18** (1989), pp. 1149–1178.