CS294 Partition Functions **Fall 2023**

Lecture Note 12

Instructor: Alistair Sinclair

Disclaimer: *These notes have not been subjected to the usual scrutiny accorded to formal publications. They may be distributed outside this class only with the permission of the Instructor.*

Introduction

In this lecture we make use of the flow-encoding technology from the last lecture to bound the mixing time of a natural Markov chain on matchings in a graph, due to [JS89]. After analyzing the basic chain, we'll look at various extensions and applications, including the major application of approximating the permanent.

12.1 A Markov chain for matchings

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 $\lambda \geq 1$,[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 *monomer-dimer* model of statistical physics: in this model, vertices connected by an edge in the matching correspond to diatomic molecules (dimers), and unmatched vertices to monatomic molecules (monomers). We recall 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.

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

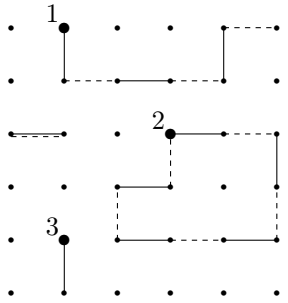

Figure 12.1: The matchings x (solid lines) and y (dotted lines).

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:

- 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

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

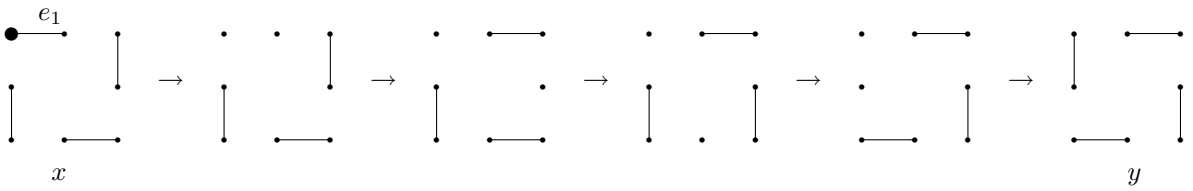

Figure 12.2: Unwinding a single cycle along the path from x to y.

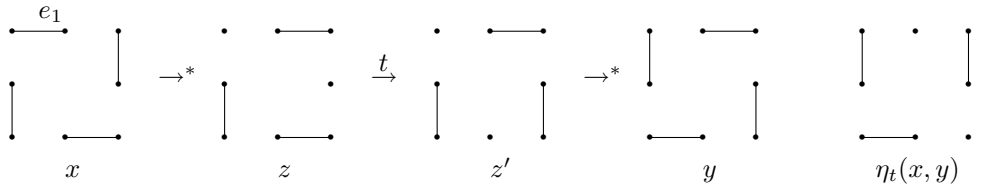

Figure 12.3: The transition t and its encoding $\eta_t(x, y)$.

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

$$\eta_t(x, y) = x \oplus y \oplus (z \cup z') \setminus \{e_1\}. \tag{12.1}$$

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 (12.1), 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 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, $\pi(x)\pi(y) \leq \lambda^2 \pi(z)\pi(\eta_t(x, y))$, so η satisfies property (ii) in the definition of encoding, with $\beta = \lambda^2$.

Analysis of τ_{mix}: For any transition $t = (z, z')$, $P(z, z') \geq \frac{1}{2\lambda|E|}$ (attained for edge deletions t). Thus by the above encoding and Claim 11.4 of the previous lecture,

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

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 $\ell(f) \leq |V|$. By Theorem 11.1, the Poincaré constant α of P satisfies

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

Then Corollary 11.2 implies that

$$\tau_x(\varepsilon) = O\left((\log(4\pi(x))^{-1} + \log\varepsilon^{-1}) \cdot \frac{1}{\alpha}\right) = O\left(\lambda^3|E||V|(\log(4\pi(x))^{-1} + \log\varepsilon^{-1})\right).$$

Note that Ω is a subset of the $2^{|E|}$ subgraphs of G, so $|\Omega| \leq 2^{|E|}$. 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

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

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

$$\tau_{\text{mix}} = O\left(\lambda^3|E|^2|V|\right),$$

which is polynomial in λ and in the size of the graph.

12.2 Extensions and applications

12.2.1 Estimating the partition function $Z(\lambda)$

Let's first confirm that the above polynomial time sampling algorithm for matchings allows us to approximate the partition function $Z(\lambda) = \sum_k m_k\lambda^k$, where m_k is the number of k-edge matchings in G. Recall that computing $Z(\lambda)$ exactly for any fixed value of $\lambda > 0$ is #P-complete, so the best we can hope for is an fpras.

We follow one of the paths outlined in our discussion of reductions from counting to sampling in an earlier lecture. 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 $\lambda_r = \lambda \leq (1+\frac{1}{n})\lambda_{r-1}$. 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:

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

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

Proof. Exercise (following discussion in earlier lecture). □

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 $(1 \pm \varepsilon)$ is $O(n^2\varepsilon^{-2})$. Also, by our mixing time analysis, the time to obtain each sample is bounded by $\text{poly}(n, \lambda)$.

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

12.2.2 Estimating the Coefficients m_k

The coefficient m_k represents the number of k-matchings in the graph. When $|V| = 2n$ is even, m_n is the number of perfect matchings. If the graph G is bipartite with n vertices on each side, counting the number of perfect matchings is equivalent to computing the *permanent* of the adjacency matrix A_G of G.

Definition 12.2. *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 $\{1, \ldots, n\}$.

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 12.3. *The sequence $\{m_k\}$ is* log-concave*: i.e., $m_{k-1}m_{k+1} \leq m_k^2$ 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 previous section.]

Claim 12.4. *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 $1 \le k' \le k$ 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 the 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 $1 \le k' \le k$.

Claim 12.4 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 $k' \le k$. By our Markov chain analysis from the previous section, 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 $(1 \pm \varepsilon)$ estimate of m_k is polynomial in n and ε^{-1}. Hence the entire algorithm takes time $\mathrm{poly}(n, m_{k-1}/m_k, \varepsilon^{-1})$ to get a $(1 \pm \varepsilon)$ 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 $\mathrm{poly}(n)$.

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 $m_{(1-\varepsilon)n}$ in time $n^{O(1/\varepsilon)}$.
(ii) Show how to use the above algorithm to find a matching of size at least $(1-\varepsilon)k^*$ in any given graph, where k^* is the (unknown) size of a maximum matching in the graph. Your algorithm should run in time $n^{O(1/\varepsilon)}$.

12.3 Approximating the permanent

Let us now focus on the important problem of counting *perfect* matchings, which as we have noted, in the case of bipartite graphs, 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 $\mathbb{Z}^d$ with periodic boundaries, 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). (**Exercise:** Check this!) Thus $m_{n-1}/m_n \ge 2^{(n-1)/2}$.

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.

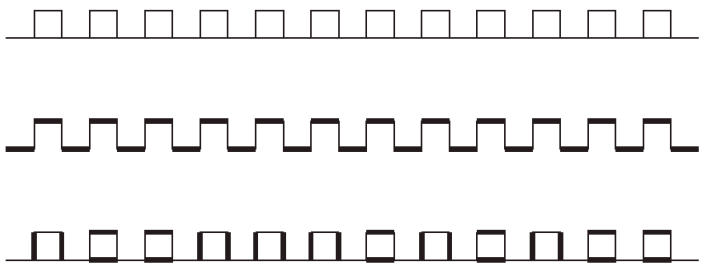

Figure 12.4: The first row shows a graph; the second row shows its unique perfect matching; the third row shows one of many near-perfect matchings.

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 essentially the same flow-encoding analysis 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., $k = n$).

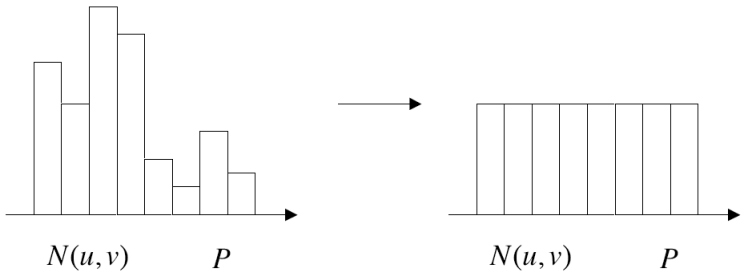

Figure 12.5: The histogram of $\{|N(u,v)|\}$ and $|P|$ before and after reweighting.

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 12.5. Note that in a "bad" graph at least one of the $|N(u,v)|$ is exponentially larger than $|P|$.

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

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

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 12.5. 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 $\Omega(\frac{1}{n^2})$ 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).

Property (ii) is not so obvious, but follows from a flow encoding argument similar to the one we used for the all-matchings chain with some more technicalities involving the hole weights: it turns out that the effect of the hole weights is indeed to effectively cancel the factor m_{n-1}/m_n from the mixing time, as we wanted, but this relies on a combinatorial inequality relating perfect and near-perfect matchings that holds only in *bipartite* graphs. Thus the algorithm we present here works only for counting perfect matchings in bipartite graphs, which corresponds to the important case of the permanent. The question of approximately counting perfect matchings in general graphs remains open (and indeed the algorithm given here is known definitely not to work, even with an arbitrarily sophisticated hole weighting scheme [ŠVW18]).

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)—i.e., we start with the complete bipartite graph $K_{n,n}$. 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 $\pi_\lambda(M) \propto \lambda(M) w_\lambda(u,v)$ for $M \in N(u,v)$, and $\pi_\lambda(M) \propto \lambda(M)$ for $M \in P$.

- 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 12.5. 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 (very close approximations to) 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 very accurately

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.

Here's a sketch of this bootstrapping procedure for estimating the new hole weights. Suppose the new activities are $\{\lambda_e\}$, so that the correct hole weights are

$$w_\lambda(u,v) = \frac{\lambda(P)}{\lambda(N(u,v))}. \tag{12.2}$$

However, we are actually sampling from a distribution π whose hole weights $w'(u,v)$ were derived from the previous activities; by definition of the current Markov chain, this distribution satisfies

$$\pi(N(u,v)) = \frac{\lambda(N(u,v))w'(u,v)}{Z}; \qquad \pi(P) = \frac{\lambda(P)}{Z},$$

where Z is the current partition function, and hence

$$w'(u,v) = \frac{\pi(N(u,v))}{\lambda(N(u,v))} \times \frac{\lambda(P)}{\pi(P)}. \tag{12.3}$$

Combining (12.2) and (12.3) we get

$$w_\lambda(u,v) = w'(u,v) \times \frac{\pi(P)}{\pi(N(u,v))}. \tag{12.4}$$

But now for every pair (u,v) we can get hold of all three quantities on the right-hand side of (12.4): the $w'(u,v)$ are just the current hole weights, which we know; and both $\pi(P)$ and $\pi(N(u,v))$ can be estimated to arbitrary accuracy by sampling from the distribution π using the current Markov chain.

Note: The above scheme extends without further modification to an fpras for the permanent of a matrix A with arbitrary non-negative entries (not just 0-1 entries, which corresponds to counting perfect matchings). General non-negative entries just correspond to edge weights in the bipartite graph, and the above scheme already counts perfect matchings with weights.

References

[JS89] M. Jerrum and A. Sinclair. Approximating the permanent. *SIAM Journal on Computing*, 18:1149–1178, 1989.

[JSV04] 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*, 51:671–697, 2004.

[ŠVW18] D. Štefankovič, E. Vigoda, and J. Wilmes. On counting perfect matchings in general graphs. *Proceedings of Latin American Symposium in Theoretical Informatics (LATIN)*, pages 873–885, 2018.