8
Graph Matchings II: Algebraic Algorithms

We now introduce some algebraic methods to find perfect matchings in general graphs. We use so-called the "polynomial method", based on the elementary fact that low-degree polynomials have few zeroes. This is a powerful and versatile idea, using a combination of basic algebra and randomness, that can be used to solve many related problems as well. For instance, we will use it to get parallel (randomized) algorithms for perfect matchings, and also to find *red-Blue perfect matchings*, an algorithm for which we know no deterministic algorithms. But before we digress to these problems, let us discuss some of the algebraic results for perfect matchings.

We focus on perfect matchings here; it is an exercise to reduce finding maximum matchings to perfect matchings.

- The first result along these lines is that of Laci Lovász, who introduced the general idea, and gave a randomized algorithm to detect the presence of perfect matchings in time $O(n^\omega)$, and to find it in time $O(mn^\omega)$. We will present all the details of this elegant idea soon.

  Lovász (1979)

- Dick Karp, Eli Upfal, and Avi Wigderson, and then Ketan Mulmuley, Umesh Vazirani, and Vijay Vazirani showed how to find such a matching in parallel. The question of getting a deterministic parallel algorithm remains an outstanding open problem, despite recent progress (which discuss at the end of the chapter).

  Karp, Upfal, and Wigderson (1986)
  Mulmuley, Vazirani, and Vazirani (1987)

- Michael Rabin and Vijay Vazirani sped up the sequential algorithm to run in $O(n \cdot n^\omega)$. This was substantially improved by the work of Marcin Mucha and Piotr Sankowski to get a runtime of $O(n^\omega)$.

  Rabin and Vazirani (1989)
  Mucha and Sankowski (2006)

8.1 Preliminaries: roots of low degree polynomials

For the rest of this lecture, we fix a field $\mathbb{F}$, and consider (univariate and multivariate) polynomials over this field. We assume that we can perform basic arithmetic operations in constant time, though sometimes it will be important to look more closely at this assumption.

For finite fields $\mathbb{F}_q$ (where q is a prime power), we can perform arithmetic operations (addition, multiplication, division) in time poly $\log q$.

Given $p(x)$, a *root/zero* of this polynomial is some value z such that $p(z)$ evaluates to zero. The critical idea for today's lecture is simple: *low-degree polynomials have "few" roots*. In this section, we will see this for both univariate and multivariate polynomials, for the right notion of "few". The following theorem well-knwon is for univariate polynomials. (The proof is essentially by induction on the degree; will add a reference.)

Theorem 8.1 (Univariate Few-Roots Theorem)**.** *A univariate polynomial $p(x)$ of degree at most d over any field $\mathbb{F}$ has at most d roots, unless $p(x)$ is zero polynomial.*

Now, for multivariate polynomials, the trivial extension of this theorem is not true. For example, $p(x,y) := xy$ has degree two, and the solutions to $p(x,y) = 0$ over the reals are exactly the points in $\{(x,y) \in \mathbb{R}^2 : \ x = 0 \text{ or } y = 0\}$, which is infinite. However, the roots are still "few", in the sense that the set of roots is very sparse in $\mathbb{R}^2$. To formalize this observation, let us write a trivial corollary of Theorem 8.1:

Corollary 8.2. *Given a non-zero univariate polynomial $p(x)$ over a field $\mathbb{F}$, such that p has degree at most d. Suppose we choose R uniformly at random from a subset $S \subseteq \mathbb{F}$. Then*

$$\Pr[p(R) = 0] \leq \frac{d}{|S|}.$$

This statement holds for multivariate polynomials as well, as we see next. The result is called the *Schwartz-Zippel lemma*, and it appears in papers by Richard DeMillo and Richard Lipton, by Richard Zippel, and by Jacob Schwartz.

R.A. DeMillo and R.J. Lipton (1978)
Zippel (1979)
Schwartz (1980)
Like many powerful ideas, the provenance of this result gets complicated. A version of this for finite fields was apparently already proved in 1922 by Øystein Ore. Anyone have a copy of that paper?

Theorem 8.3. *Let $p(x_1, \ldots, x_n)$ be a non-zero polynomial over a field $\mathbb{F}$, such that p has degree at most d. Suppose we choose values $R_1, \ldots, R_n$ independently and uniformly at random from a subset $S \subseteq \mathbb{F}$. Then*

$$\Pr[p(R_1, \ldots, R_n) = 0] \leq \frac{d}{|S|}.$$

Hence, the number of roots of p inside S^n is at most $d|S|^{n-1}$.

Proof. We argue by induction on n. The base case of $n = 1$ considers univariate polynomials, so the claim follows from Theorem 8.1. Now for the inductive step for n variables. Let k be the highest power of x_n that appears in p, and let q be the quotient and r be the remainder when dividing p by x_n^k. That is, let $q(x_1, \ldots, x_{n-1})$ and $r(x_1, \ldots, x_n)$ be the (unique) polynomials such that

$$p(x_1, \ldots, x_n) = x_n^k \, q(x_1, \ldots, x_{n-1}) + r(x_1, \ldots, x_n),$$

A *monomial* is a product a collection of variables. The degree of a monomial is the sum of degrees of the variables in it. The degree of a polynomial is the maximum degree of any monomial in it.

where the highest power of x_n in r is less than k. Now letting $\mathcal{E}$ be the event that $q(R_1, \ldots, R_{n-1})$ is zero, we find

$$\begin{aligned}\Pr[p(R_1, \ldots, R_n) = 0] &= \Pr[p(R_1, \ldots, R_n) = 0 \mid E] \Pr[\mathcal{E}] \\ &\quad + \Pr\left[p(R_1, \ldots, R_n) = 0 \mid \overline{\mathcal{E}}\right] \Pr\left[\overline{\mathcal{E}}\right] \\ &\leq \Pr[\mathcal{E}] + \Pr\left[p(R_1, \ldots, R_n) = 0 \mid \overline{\mathcal{E}}\right]\end{aligned}$$

By the inductive assumption, and noting that q has degree at most $d - k$, we know

$$\Pr[\mathcal{E}] = \Pr[q(R_1, \ldots, R_{n-1}) = 0] \leq (d-k)/|S|.$$

Similarly, fixing the values of $R_1, \ldots, R_{n-1}$ and viewing p as a polynomial only in variable x_n (with degree k), we know

$$\Pr\left[p(R_1, \ldots, R_n) = 0 \mid \overline{\mathcal{E}}\right] \leq k/|S|.$$

Thus we get

$$\Pr[p(R_1, \ldots, R_n) = 0] \leq \frac{d-k}{|S|} + \frac{k}{|S|} = \frac{d}{|S|}. \qquad \square$$

Remark 8.4. Finding the set $S \subseteq \mathbb{F}$ such that $|S| \geq dn^2$, guarantees that if p is a non-zero polynomial,

$$\Pr[p(R_1, \ldots, R_n) = 0] \leq \frac{1}{n^2}.$$

Naturally, if p is zero polynomial, then the probability equals 1.

8.2 *Detecting Perfect Matchings by Computing a Determinant*

Let us solve the easier problem of detecting a perfect matching in a graph, first for bipartite graphs, and then for general graphs. We define the *Edmonds matrix* of a bipartite graph G.

Definition 8.5. For a bipartite graph $G = (L, R, E)$ with $|L| = |R| = n$, its *Edmonds matrix* $\mathbf{E}(G)$ is the following $n \times n$ matrix of indeterminates/variables.

$$\mathbf{E}_{i,j} = \begin{cases} 0 & \text{if } (i,j) \notin E \text{ and } i \in L, j \in R \\ x_{i,j} & \text{if } (i,j) \in E \text{ and } i \in L, j \in R. \end{cases}$$

Example 8.6. The Edmonds matrix of the graph to the right is

$$\mathbf{E} = \begin{bmatrix} x_{11} & x_{12} \\ 0 & x_{22} \end{bmatrix},$$

which has determinant $x_{11}x_{22}$.

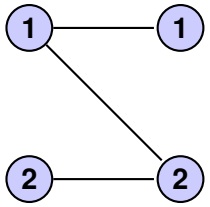

Figure 8.1: Bipartite graph

Recall the Leibniz formula to compute the determinant, and apply it to the Edmonds matrix:

$$\det(\mathbf{E}(G)) = \sum_{\sigma \in S_n} (-1)^{\text{sign}(\sigma)} \prod_{i=1}^{n} \mathbf{E}_{i,\sigma(i)}$$

There is a natural correspondence between potential perfect matchings in G and permutations $\sigma \in S_n$, where we match each vertex $i \in L$ to vertex $\sigma(i) \in R$. Moreover, the term in the above expansion corresponding to a permutation σ gives a non-zero *monomial* (a product of x_{ij} variables) if and only if all the edges coresponding to that permutation exist in G. Moreover, all the monomials are distinct, by construction. This proves the following simple claim.

Proposition 8.7. *Let $\mathbf{E}(G)$ denote the Edmonds matrix of a bipartite graph G. Then $\det(\mathbf{E}(G))$ is a non-zero polynomial (over any field $\mathbb{F}$) if and only if G contains a perfect matching.*

However, writing out this determinant of indeterminates could take exponential time—it would correspond to a brute-forece check of all possible perfect matchings. Lovász's idea was to use the randomized algorithm implicit in Theorem 8.3 to *test* whether G contains a perfect matching.

Algorithm 10: PM-tester(bipartite graph G, $S \subseteq \mathbb{F}$)

```
10.1 E ← Edmonds matrix for graph G
10.2 For each non-zero entry E_ij, sample R_ij ∈ S independently and
       uniformly at random
10.3 Ẽ ← E({R_ij}_{i,j}) be matrix with sampled values substituted
10.4 if det(Ẽ) = 0 then
10.5 |   return G does not have a perfect matching (No)
10.6 else
10.7 |   return G contains a perfect matching (Yes)
```

Lemma 8.8. *For $|S| \geq n^3$, Algorithm 10 always returns `No` if G has no perfect matching, while it says `Yes` with probability at least $1 - \frac{1}{n^2}$ otherwise. Moreover, the algorithm can be implemented in time $O(n^\omega)$, where ω is the exponent of matrix multiplication.*

Proof. The success probability follows from Remark 8.4, and the fact that the determinant is a polynomial of degree n. Assuming that arithmetic operations can be done in constant time, we can compute the determinant of $\widetilde{\mathbf{E}}$ in time $O(n^3)$, using Gaussian elimination. Hence Algorithm 10 easuly runs in time $O(n^3)$. In fact, Bunch and Hopcroft proved that both computing matrix inverses and determinants can be done in asymptotically the same time as matrix multiplication. Thus, we can make Algorithm 10 run in time $O(n^\omega)$. □

If we work over finite fields, the size of the numbers is not an issue. However, Gaussian Elimination over the rationals could cause some of the numbers to get unreasonably large. Ensuring that the numbers remain polynomially bounded requires care; see Edmonds' paper, or a book on numerical methods.
J.R. Bunch and J.E. Hopcroft (1974)

8.2.1 *Non-bipartite matching*

The extension to the non-bipartite case requires a very small change: instead of using the Edmonds matrix, which is designed for bipartite graphs, we use the analogous object for general graphs. This object was introduced by Bill Tutte in his 1947 paper with the Tutte-Berge theorem.

Tutte (1947)

Definition 8.9. For a general graph $G = (V, E)$ with $|V| = n$, the *Tutte matrix* $\mathbf{T}(G)$ of G is the $n \times n$ skew-symmetric matrix given by

A matrix A is *skew-symmetric* if $A^\intercal = -A$.

$$\mathbf{T}_{i,j} = \begin{cases} 0 & \text{if } (i,j) \notin E \text{ or } i = j \\ x_{i,j} & \text{if } (i,j) \in E \text{ and } i < j \\ -x_{j,i} & \text{if } (i,j) \in E \text{ and } i > j. \end{cases}$$

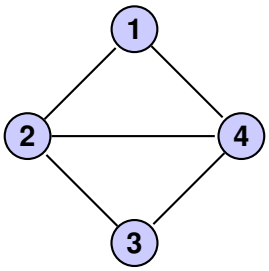

Figure 8.2: Non-bipartite graph

Example 8.10. For the graph to the right, the Tutte matrix is

$$\begin{bmatrix} 0 & x_{1,2} & 0 & x_{1,4} \\ -x_{1,2} & 0 & x_{2,3} & x_{2,4} \\ 0 & -x_{2,3} & 0 & x_{3,4} \\ -x_{1,4} & -x_{2,4} & -x_{3,4} & 0 \end{bmatrix}$$

And its determinant is blah blah.

Observe that now each variable occurs twice, with the variables below the diagonal being the negations of those above. We claim the same property for this matrix as we did for the Edmonds matrix:

Theorem 8.11. *For any graph G, the determinant of the Tutte matrix $\mathbf{T}(G)$ is a non-zero polynomial over any field $\mathbb{F}$ if and only if there exists a perfect matching in G.*

Proof. As before, the determinant is

$$\det(\mathbf{T}(G)) = \sum_{\sigma} (-1)^{\text{sign}(\sigma)} \prod_{i=1}^{n} \mathbf{T}_{i,\sigma(i)}.$$

One direction of the theorem is easy: if G has a perfect matching M, consider the permutation σ mapping each vertex to the other endpoint of the matching edge containing it. The corresponding monomial above is $\pm \prod_{e \in M} x_e^2$, which cannot be cancelled by any other permutation, and makes the determinant non-zero over any field.

To prove the converse, suppose the determinant is non-zero. In the monomial corresponding to permutation σ, either each $(i, \sigma(i))$ in the product corresponds to an edge of G, or else the monomial is zero. This means each non-zero monomial of $\det(\mathbf{T}(G))$ chooses an edge incident to i, for each vertex $i \in [n]$, giving us a *cycle cover* of G.

If the cycle cover for σ has an odd-length cycle, take the permutation σ' obtained by reversing the order of, say, the first odd cycle in it. This does not change the sign of the permutation, which depends on how many odd and even cycles there are, but the skew symmetry and the odd cycle length means the product of matrix entries flips sign. Hence the monomials for σ and σ' cancel each other. Formally, we get a bijection between permutations with odd-length cycles that cancel out.

The remaining monomials corresponding to cycle covers with even cycles. Choosing either the odd edges or even edges on each such even cycle gives a perfect matching. □

Now given Theorem 8.11, the Tutte matrix can simply be substituted instead of the Edmonds matrix to extend the results to general graphs.

8.3 From Detecting to Finding Perfect Matchings

We can convert the above perfect matching tester (which solves the *decision version* of the perfect matching problem) into an algorithm for the *search version*: one that outputs a perfect matching in a graph (if one exists), using the simple but brilliant idea of *self-reducibility*. Suppose that graph G has a perfect matching. Then we can pick any edge $e = uv$ and check if $G[E - e]$, the subgraph of G obtained by dropping just the edge e, contains a perfect matching. If not, then edge e must be part of every perfect matching in G, and hence we can find a perfect matching on the induced subgraph $G[V \setminus \{u, v\}]$. The following algorithm is based on this observation.

We are reducing the problem to smaller instances of itself, hence the name.

Algorithm 11: Find-PM(bipartite graph G, $S \subseteq \mathbb{F}$)

11.1 **Assume:** G has a perfect matching **let** $e = uv$ be an edge in G **if** *PM-tester*$(G[E - e], S)$ **==** **Yes** **then**
11.2 | **return** Find-PM$(G[E - e], S)$
11.3 **else**
11.4 | $M' \leftarrow$ Find-PM$(G[V - \{u, v\}], S)$
11.5 | **return** $M' \cup \{e\}$

Theorem 8.12. *Let $|S| \geq n^3$. Given a bipartite graph G that contains some perfect matching, Algorithm 11 finds a perfect matching with probability at least $\frac{1}{2}$, and runs in time $O(m \cdot n^\omega)$.*

Proof. At each step, we call the tester once, and then recurse after either deleting an edge or two vertices. Thus, the number of total recursive steps inside Algorithm 11 is at most $\max\{m, n/2\} = m$, if the graph is connected. This gives a runtime of $O(m \cdot n^\omega)$. Moreover,

at each step, the probability that the tester returns a wrong answer is at most $\frac{1}{n^2}$, so the PM-tester makes a mistake with probability at most $\frac{m}{n^2} \leq 1/2$, by a union bound. □

Observe that the algorithm assumes that G contains a perfect matching. We could simply run the PM-tester once on G at the beginning to check for a perfect matching, and then proceed as above. Or indeed, we could just run the algorithm regardless; if G has no perfect matching, there is no danger of this algorithm erronenously returning one.

Moreover, there are at least two ways reducing the error probability to $1/n^c$: we could increase the size of S to n^{s+3}, or we could repeat the above algorithm $c \log_2 n$ times. For the latter approach, the probability of not getting a perfect matching in all iterations is at most $(1/2)^{c \log_2 n} = \frac{1}{n^c}$. Hence we get correctness with high probability.

Corollary 8.13. *Given a bipartite graph G containing a perfect matching, there is an $O(m \cdot n^\omega \log n)$-time algorithm which finds a perfect matching with high probability.*

Exercise 8.14. Reduce the time-complexity of the basic version of Algorithm 11 from $O(m \cdot n^\omega)$ to $O(n \log n \cdot n^\omega)$.

8.3.1 The Algorithm of Rabin and Vazirani

Rewrite this section. How can we speed up the algorithm further? The improvement we give here is small, it only removes a logarithmic term from the algorithm you get from Exercise 8.14, but it has a nice idea that we want to emphasize. Again, we only focus on the bipartite case, and leave the general case for the reader. Also, we identify the nodes of both L and R with $[n]$, so that we can index the rows and columns of the Edmonds matrix using vertices in L and R respectively.

We can view Algorithm 11 as searching for a permutation π such that $M := \{i\pi(i) \mid i \in [n]\}$ is a perfect matching in G. Hence it picks a vertex, say $1 \in L$, and searches for a vertex $j \in R$ such that there is an edge $1j$ covering vertex 1, and also the remaining graph has a perfect matching. Interestingly, the remaining graph has an Edmonds matrix which is simple: it is simply the matrix $\mathbf{E}_{-1,-j}$, which is our notation from dropping row 1 and column j from $\mathbf{E}$.

Therefore, our task is simple: find a value j such that $1j$ is an edge in G, and also $\det(\mathbf{E}_{-1,-j})$ is a non-zero polynomial. Doing this naively would require n determinant computations, one for each j, and we'd be back to square one. But the smart observation is to recall Cramer's rule for the inverse for any matrix A:

Some jargon that may be useful:

$$A^{-1} = \frac{\text{adjugate}(A)}{\det(A)},$$

where adjugate(A) is the transpose of the *cofactor matrix* of A, given by

$$\text{cofactor}(A)_{p,q} := (-1)^{p+q} \det(A_{-p,-q}),$$

where $\det(A_{-p,-q})$ is also called a *minor* of A.

$$\left(A^{-1}\right)_{i,j} = \frac{(-1)^{i+j}\det(A_{-j,-i})}{\det(A)}. \tag{8.1}$$

Take the matrix $\widetilde{\mathbf{E}}$ obtained by substituting random values into the Edmonds matrix $\mathbf{E}$, and assume the set S and field $\mathbb{F}$ are of size at least n^{10}, say, so that all error probabilities are tiny. Compute its inverse in time $O(n^\omega)$, and use (8.1) to get all the values $\det(\widetilde{\mathbf{E}}_{-1,-j})$:

$$\det(\widetilde{\mathbf{E}}_{-1,-j}) = \left(\widetilde{\mathbf{E}}^{-1}\right)_{j,1} \times (-1)^{j+1} \times \det(\widetilde{\mathbf{E}})$$

using just n scalar multiplications. In fact, it suffices to find a non-zero $\widetilde{\mathbf{E}}^{-1})_{j,1}$, which indicates that $\det(\widetilde{\mathbf{E}}_{-1,-j})$ is non-zero, and hence the corresponding $\det(\mathbf{E}_{-1,-j})$ (without the tilde) is a non-zero polynomial, so that $G[V \setminus \{1, j\}]$ has a perfect matching.

In summary, by computing one matrix inverse and n scalar multiplications, we can figure out one edge of the matching. Hence the runtime can be made $O(n \cdot n^\omega)$. Extending this to general graphs requires a bit more work; we refer to the Rabin and Vazirani paper for details. Also, Marcin Mucha's thesis has a very well-written introduction which discusses these details, and also gives the details of his improvement (with Sankowski) to $O(n^\omega)$ time.

8.3.2 The Polynomial Identity Testing (PIT) Problem

In *Polynomial Identity Testing* we are given a polynomial $P(x_1, x_2, \ldots, x_n)$ over some field $\mathbb{F}$, and we want to test if it is identically zero or not. If P were written out explicitly as a list of monomials and their coefficients, this would not be a problem, since we could just check that all the coefficients are zero. But if P is represented implicitly, say as a determinant, then things get more tricky. A big question is whether polynomial identity testing (PIT) can be derandomized.

We don't know deterministic algorithms that given P can decide whether P is identically zero or not, in poly-time. How is P given, for this question? Even if P is given as an arithmetic circuit (a circuit whose gates are addition, subtraction and multiplication, and inputs are the variables and constants), it turns out that derandomizing PIT will result in surprising circuit lower bounds—for example, via a result of Kabanets and Impagliazzo. Derandomizing special cases of PIT can be done. For example, just the PIT instances that come from matchings can, however, be derandomized. This was shown in work by Jim Geelen and Nick Harvey, among others; however, but the runtime seems to get much worse.

J.F. Geelen (2000)
N.J.A. Harvey (2009)

8.4 Red-Blue Perfect Matchings

To illustrate the power of the algebraic approach, let us now consider the *red-blue matching* problem. We solve this problem using the algebraic approach and randomization. Interestingly, no determistic polynomial-time algorithm is currently known for this problem! Again, we consider bipartite graphs for simplicity.

Given a graph G where each edge is colored either red or blue, and an integer k, a k-red matching is a perfect matching in G which has exactly k red edges. The goal of the red-blue matching problem is to decide whether G has a k-red matching. (We focus on the decision version of the problem; the self-reducibility ideas used above can solve the search version.

To begin, let's solve the case when G has a *unique* red-blue matching with k red edges. Define the following $n \times n$ matrix:

$$\mathbf{M}_{i,j} = \begin{cases} 0 & \text{if } (i,j) \notin E, \\ 1 & \text{if } (i,j) \in E \text{ and colored blue,} \\ y & \text{if } (i,j) \in E \text{ and colored red.} \end{cases}$$

Claim 8.15. Let G have at most one perfect matching with k red edges. The determinant $\det(\mathbf{M})$ has a term of the form $c_k y^k$ if and only if G has a k-red matching.

Proof. Consider $p(y) := \det(\mathbf{M})$ as a univariate polynomial in the variable y. Again, using the Leibniz formula, the only way to get a non-zero term of the form $c_k y^k$ is if the graph has a k-red matching. And since we assumed that G has at most one such perfect matching, such a term cannot be cancelled out by other such matchings. □

The polynomial $p(y)$ has degree at most n, and hence we can recover it by Lagrangian interpolation. Indeed, we can choose $n+1$ distinct numbers $a_0, \ldots, a_n$, and evaluate $p(a_0), \ldots, p(a_n)$ by computing the determinant $\det(\mathbf{M})$ at $y = a_i$, for each i. These $n+1$ values are enough to determine the polynomial as follows:

$$p(y) = \sum_{i=1}^{n+1} p(a_i) \prod_{j \neq i} \left(\frac{x - a_j}{a_i - a_j} \right).$$

(E.g., see 451 lecture notes or Ryan's lecture notes.) Note this is a completely deterministic algorithm, so far.

8.4.1 Getting Rid of the Uniqueness Assumption

To extend to the case where G could have many k-red matchings, we can redefine the matrix as the following:

$$\mathbf{M}_{i,j} = \begin{cases} 0 & \text{if } (i,j) \notin E, \\ x_{ij} & \text{if } (i,j) \in E \text{ and colored blue,} \\ yx_{ij} & \text{if } (i,j) \in E \text{ and colored red.} \end{cases}$$

The determinant $\det(\mathbf{M})$ is now a polynomial in $m+1$ variables and degree at most $2n$. Writing

$$P(\mathbf{x}, y) = \sum_{i=0}^{n} y^i Q_i(\mathbf{x}),$$

where Q_i is a multilinear degree-n polynomial that corresponds to all the i-red matchings. If we set the $\mathbf{x}$ variables randomly (say, to values $x_{ij} = a_{ij}$) from a large enough set S, we get a polynomial $R(y) = P(\mathbf{a}, y)$ whose only variable is y. The coefficient of y^k in this polynomial is $Q_k(\mathbf{a})$, which is non-zero with high probability, by the Schwartz-Zippel lemma. Now we can again use interpolation to find out this coefficient, and decide the red-blue matching problem based on whether it is non-zero.

Multilinear just means that the degree of each variable in each monomial is at most one.

8.5 Matchings in Parallel, and the Isolation Lemma

One of the "killer applications" of the algebraic method for finding a perfect matching is that the approach extends to getting a (randomized) parallel algorithm as well. The basic idea is simple when there is a unique perfect matching. Indeed, computing the determinant can be done in parallel with poly-logarithmic depth and polynomial work. Hence, for each edge e we can run the PM-tester algorithm on G, and also on $G[E-e]$ to see if e belongs to this unique perfect matching; we output e if it does.

However, this approach fails when the graph has multiple perfect matchings. A fix for this problem was given by Mulmuley, Vazirani, and Vazirani by adding further randomness! The approach is first extend the approach from §8.2 to find a *minimum-weight* perfect matching using the Tutte matrix and Schwartz-Zippel lemma, as long as the weights are polynomially bounded. (Exercise: solve this!) The trickier part is to show that assigning random polynomially-bounded weights to the edges of G causes it to have a *unique* minimum-weight perfect matching with high probability. Then this unique matching can also be found in parallel, as we outlined in the previous paragraph.

K. Mulmuley, U.V. Vazirani and V.V. Vazirani (1987)

The proof showing that random weights result in a unique minimum-weight perfect matching is via a beautiful result called the *Isolation Lemma*. Let us give its simple elegant proof.

Theorem 8.16. *Consider a collection $\mathcal{F} = \{M_1, M_2, \dots, \}$ of sets over a universe E of size m. Assign a random weight to each elements of E, where the weights are drawn independently and uniformly from $\{1, \dots, 2m\}$. Then there exists a unique minimum-weight set with probability at least $\frac{1}{2}$.*

Proof. Call an element $e \in E$ "confused" if the weight of a minimum-weight set containing e is the same as the weight of a minimum-weight set *not* containing e. We claim that any specific element e is confused with probability at most $1/2m$. Observe is that there exists a confused element if and only if there are two minimum-weight sets, so using the claim and taking a union bound over all elements proves the theorem.

To prove the claim, make the random choices for all elements except e. Now the identity (and weight) of the minimum-weight set not containing e is determined; let its weight be W^-. Also, the identity (but *not* the weight) of the minimum-weight set containing e is determined. Its weight is not determined because the random choice for e's weight has not been made, so denote its weight by $W^+ + w_e$, where w_e is the random variable denoting the weight of e. Now e will be confused precisely if $W^- = W^+ + w_e$, i.e., if $w_e = W^- - W^+$. But since w_e is chosen uniformly at random from a set of size $2m$, this probability is at most $1/2m$, as claimed. □

We are using the *principle of deferred decisions* again.

It is remarkable the result does not depend on number of sets in $\mathcal{F}$, but only on the size of the universe. We also emphasize that the weights being drawn from a polynomially-sized set is what gives the claim its power: it is trivial to obtain a unique minimum-weight set if the weights are allowed to be in $\{1, \dots, 2^m\}$. (Exercise: how?) Finally, the proof strategy for the Isolation Lemma is versatile and worth remembering.

8.5.1 Towards Deterministic Algorithms

The question of finding perfect matchings *deterministically* in poly-logarithmic depth and polynomial work still remains open. Some recent work of Fenner, Gurjar, and Thierauf, and of Svensson and Tarnawski has shown how to obtain poly-logarithmic depth and quasi-polynomial work; we may see some of these ideas in an upcoming HW.

S. Fenner, R. Gurjar, and T. Thierauf (2016)
O. Svensson and J. Tarnawski (2017)

8.6 A Matrix Scaling Approach

A very different matrix-based approach to finding (fractional) matchings uses the concept called *matrix scaling*. The idea very elegant. Given a non-negative matrix A, and two non-negative diagonal matrices R and C, consider the *scaled matrix*

$$B := RAC.$$

In other words, taking the matrix A, and scaling each row i by R_{ii} and each column j by C_{jj}, gives the matrix B.

Theorem 8.17. *A bipartite graph G admits a perfect matching if and only if for each $\varepsilon > 0$ there exist non-negative matrices R, C such that RA_GC is ε-approximate doubly-stochastic.*

Proof. To come soon □

A matrix A is *doubly-stochastic* if it has unit row- and column-sums. In other words, $A\mathbb{1} = A^\intercal\mathbb{1} = \mathbb{1}$. The ε-doubly-stochasticity requires that $A\mathbb{1}, A^\intercal\mathbb{1}$ both have entries in $(1-\varepsilon, 1+\varepsilon)$.

Given the adjacency matrix $A \in \{0,1\}^n$ for the bipartite graph G, we now try to find scaling matrices R and C. Since we want the row- and column-sums to be close to 1, one "greedy" idea is to start with A and repeatedly do the following two steps:

1. Scale each row to make the row sums equal to 1; this may put the column sums out of whack.
2. Scale each column to make the row sums equal to 1; this may now mess up the row sums.

We show that if we ever reach a matrix where both row and column sums are very close to 1, then Theorem 8.17 tells us that the graph has a perfect matching. And if we don't manage to get close to 1 in a "reasonable" time (which depends on n and ε), interestingly we can conclude it has no perfect matching!

To make this precise, let's define two diagonal matrices $R(A) := \text{diag}(A\mathbb{1})$ and $C(A) = \text{diag}(A^\intercal\mathbb{1})$. Then the algorithm becomes:

Algorithm 12: Sinkhorn Scaling(A)

```
12.1 for i = 1, 2, ..., T do
12.2     A ← R(A)^{-1} A
12.3     if ||I − C(A)||_2 ≤ 1/√n then return true
12.4     A ← A C(A)^{-1}
12.5     if ||R(A) − I||_2 ≤ 1/√n then return true
12.6 return false
```

Theorem 8.18. *If $T \geq \text{poly}(n)$ then Algorithm 12 outputs `true` if and only if the bipartite graph G has a perfect matching.*

Proof. To come soon □