CS 598CCI: Topics in Graph Algorithms **Lecture date: 29 January, 2020**
Instructor: Chandra Chekuri **Scribes: Tin-Yin Lai and Robert Andrews**

1 Matchings via Determinants

Let $G = (V, E)$ be an undirected graph. A subset of edges $M \subseteq E$ is a *matching* in G if no two edges in M share a vertex. We say that M is a *perfect matching* if every vertex of G is incident to some edge of M.

Matchings have a number of important applications in computer science and possess interesting mathematical and structural properties. Given a graph G, some algorithmic questions of interest related to matchings are the following.

- Does G have a perfect matching?
- What is the size of a maximum cardinality matching in G?
- Given weights on the edges of G, what is the maximum possible weight of a matching?
- Given costs on the edges of G, what is the minimum cost of a perfect matching (if one exists)?

In bipartite graphs, most of the above questions can be answered via reductions to max-flow and min-cost flow. Recall that a graph $G = (V, E)$ is *bipartite* if the vertices V can be partitioned into two disjoint sets A and B such that every edge of G has exactly one endpoint in A and one endpoint in B.

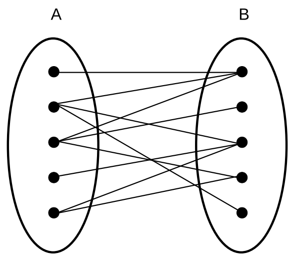


Figure 1: An example of a bipartite graph.

These techniques break for non-bipartite graphs. In the non-bipartite setting, the maximum matching problem can be solved via the blossom algorithm of Edmonds, who built on prior structural work of Tutte and Berge. In this lecture, we will tackle bipartite and non-bipartite perfect matching via a linear algebraic approach, which will yield randomized algorithms for these problems. Known combinatorial[1] algorithms for these problems are deterministic, so at first glance it appears that we do not gain much. However, these algebraic algorithms will have the extra advantage that they are easily parallelized, a feature which known combinatorial algorithms lack.

[1]By "combinatorial," we loosely mean algorithms which do not rely on fast matrix multiplication.

First, some mathematical preliminaries. For an integer $n \geq 1$, let S_n be the set of $n!$ permutations over $\{1, 2, 3, ..., n\}$. These from a group known as the *symmetric group*. For $\sigma \in S_n$, we denote by $\sigma(i)$ the element that i is mapped to by σ. To every permutation $\sigma \in S_n$, we assign a *sign*, denoted by $\operatorname{sgn}(\sigma)$, which is either $+1$ or -1. Formally,

$$\operatorname{sgn}(\sigma) := \begin{cases} 1 & \text{if the number of inversions in } \sigma \text{ is even} \\ -1 & \text{otherwise,} \end{cases}$$

where an *inversion* of σ is a pair (i, j) such that $i < j$ and $\sigma(i) > \sigma(j)$.

Let A be a $n \times n$ matrix. The *determinant* of A, denoted $\det(A)$, can be defined in several ways. We will use the fact that the determinant is a polynomial function of the entries of the matrix A, given by the formula

$$\det(A) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)}.$$

Example 1 *As an example, consider the* 2×2 *matrix* $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$. *The group* S_2 *has two elements, which we denote by* σ_1 *and* σ_2*. The permutation* σ_1 *is given by* $\sigma_1(1) = 1$ *and* $\sigma_1(2) = 2$*. The permutation* σ_2 *is given by* $\sigma_2(1) = 2$ *and* $\sigma_2(2) = 1$*. One can verify that* $\operatorname{sgn}(\sigma_1) = 1$ *and* $\operatorname{sgn}(\sigma_2) = -1$*. Using the above definition of the determinant, we see that* $\det(A)$ *coincides with the familiar formula*

$$\det(A) = A_{1,1}A_{2,2} - A_{1,2}A_{2,1} = ad - bc.$$

Closely related to the determinant of an $n \times n$ matrix is another quantity called the *permanent*, denoted by $\operatorname{per}(A)$. The permanent is given by

$$\operatorname{per}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}.$$

Note that the only difference between the expressions for $\det(A)$ and $\operatorname{per}(A)$ is the absence of the $\operatorname{sgn}(\sigma)$ term in $\operatorname{per}(A)$. Perhaps surprisingly, this makes a huge difference: there are efficient algorithms (both sequential and parallel) for computing the determinant, but the permanent is conjectured to be extremely hard to compute.

1.1 Schwartz-Zippel Lemma

An important tool in randomized algebraic algorithms is the following lemma, commonly referred to as the Schwartz-Zippel lemma, which says that a low-degree multivariate polynomial cannot vanish on too many points of a large enough grid.

Recall that the *degree* of a multivariate polynomial p is the maximum degree of the monomials appearing in p. For example, if

$$p(x_1, x_2, x_3) = 3x_1x_2 + 10x_1^2 - x_1x_2x_3^2 + x_3^2 - x_1x_2^3,$$

then $\deg(p) = 4$, since the monomials $-x_1x_2x_3^2$ and $-x_1x_2^3$ both have degree 4.

Lemma 1 *Let $\mathbb{F}$ be a field and let $p(x_1, \ldots x_n) \in \mathbb{F}[x_1, \ldots, x_n]$ be a nonzero multivariate polynomial of degree d. Let $S \subseteq \mathbb{F}$ be a finite set. Then*

$$\Pr[p(a_1, \ldots, a_n) = 0] \leq \frac{d}{|S|},$$

where $a_1, \ldots, a_n$ are chosen independently and uniformly at random from S.

Proof: We proceed by induction on n.

Base case: When $n = 1$, we know that $p(a_1) = 0$ if and only if a_1 is a root of p. The fundamental theorem of algebra says that a degree d univariate polynomial has at most d roots. Hence

$$\Pr[p(a_1) = 0] \leq \frac{d}{|S|},$$

where a_1 is chosen uniformly at random from S.

Inductive step: Suppose $n \geq 2$. Without loss of generality, the variable x_n appears in p, so we may write

$$p(x_1, \ldots, x_n) = \sum_{j=0}^{t} p_j(x_1, \ldots, x_{n-1}) x_n^j,$$

where t is the maximum degree of x_n in p and $p_0, \ldots, p_t$ are polynomials in the variables $x_1, \ldots, x_{n-1}$. In particular, $p_t(x_1, \ldots, x_{n-1}) \neq 0$, since there is some non-zero term of p in which contains x_n^t. Note that $\deg(p_t) = d - t$.

Since $a_1, \ldots, a_n$ are picked independently from S, we imagine first picking $a_2, ..., a_n$ and then picking a_1. We then have

$$\begin{aligned}
\Pr[p(a_1, \ldots, a_n) \neq 0] &\geq \Pr[p(a_1, \ldots, a_{n-1}, x_n) \neq 0] \\
&\qquad \cdot \Pr[p(a_1, \ldots, a_n) \neq 0 \mid p(a_1, \ldots, a_{n-1}, x_n) \neq 0] \\
&\geq \Pr[p_t(a_1, \ldots, a_{n-1}) \neq 0] \\
&\qquad \cdot \Pr[p(a_1, \ldots, a_n) \neq 0 \mid p(a_1, \ldots, a_{n-1}, x_n) \neq 0].
\end{aligned}$$

By induction, we have

$$\Pr[p_t(a_1, \ldots, a_{n-1}) \neq 0] \geq 1 - \frac{d-t}{|S|}$$

and

$$\Pr[p(a_1, \ldots, a_n) \neq 0 \mid p(a_1, \ldots, a_{n-1}, x_n) \neq 0] \geq 1 - \frac{t}{|S|}.$$

Combining these, we get

$$\begin{aligned}
\Pr[p(a_1, \ldots, a_n) \neq 0] &\geq \left(1 - \frac{d-t}{|S|}\right)\left(1 - \frac{t}{|S|}\right) \\
&\geq 1 - \frac{d}{|S|}.
\end{aligned}$$

Hence $\Pr[p(a_1, \ldots, a_n) = 0] \leq \frac{d}{|S|}$ as claimed. □

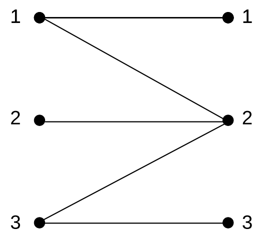
1.2 Bipartite Graphs

Suppose $G = (V_1 \cup V_2, E)$ is a bipartite graph such that $|A| = |B|$. We want to determine whether or not G has a perfect matching. We will present a randomized algorithm for this problem due to Lovász. Let $n \coloneqq |A| = |B|$. Consider the symbolic $n \times n$ matrix

$$A_{i,j} = \begin{cases} x_{i,j} & \text{if } (i,j) \in E \\ 0 & \text{otherwise,} \end{cases}$$

where $x_{i,j}$ is a variable.

Example 2 *As an example, consider the bipartite graph below. The matrix corresponding to this*



graph is given by

$$\begin{pmatrix} x_{1,1} & x_{1,2} & 0 \\ 0 & x_{2,2} & 0 \\ 0 & x_{3,2} & x_{3,3} \end{pmatrix}.$$

By definition, $\det(A)$ is a multilinear polynomial in the variables $x_{i,j}$. A polynomial is *multilinear* if every variable appears with degree at most 1 in the polynomial. The utility of this matrix A lies in the following lemma.

Claim 2 *Let $G = (V_1 \cup V_2, E)$ be a bipartite graph and let A be the symbolic matrix described above. Then G has a perfect matching if and only if $\det(A) \neq 0$ as a polynomial.*

Proof: Recall that, by definition, we have

$$\det(A) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)}.$$

First, suppose G has a perfect matching M. For $i \in V_1$, denote by $\sigma(i)$ the vertex $j \in V_2$ which is matched to i by M. Since M is a matching, σ is a permutation. Moreover, for each $i \in V_1$, the edge $(i, \sigma(i))$ is present in E. This implies that $\prod_{i=1}^{n} A_{i,\sigma(i)} = \prod_{i=1}^{n} x_{i,\sigma(i)} \neq 0$, so $\det(A) \neq 0$.

In the other direction, suppose $\det(A) \neq 0$. Then there is some $\sigma \in S_n$ such that $\prod_{i=1}^{n} A_{i,\sigma(i)} \neq 0$. This implies that $A_{i,\sigma(i)} \neq 0$ for all $i \in V_1$, so the edge $(i, \sigma(i))$ is present in G for all $i \in V_1$. Taking the union of these edges then forms a perfect matching. □

We cannot hope to expand $\det(A)$ as a sum of monomials, as simply writing down the resulting expression may take $\Omega(n!)$ time in the worst case. However, we can use the Schwartz-Zippel lemma to test if $\det(A)$ is a non-zero polynomial. Suppose we randomly assign each variable $x_{i,j}$ to an integer from $\{1, \dots, 2n\}$ and numerically compute the determinant of the resulting matrix. Let $\vec{a}$ denote the assignment of the variables and let $\det(A)|_{\vec{x}=\vec{a}}$ denote the value of $\det(A)$ under the substitution $x_{i,j} \mapsto a_{i,j}$. If $\det(A) = 0$, then clearly $\det(A)|_{\vec{x}=\vec{a}} = 0$. On the other hand, since $\det(A)$ is a polynomial of degree at most n, the Schwartz-Zippel lemma implies

$$\Pr[\det(A)|_{\vec{x}=\vec{a}} = 0 \mid \det(A) \neq 0] \leq \frac{1}{2}.$$

Hence, we obtain a randomized algorithm with error probability 1/2. As usual, we can reduce the error probability substantially by running this algorithm multiple times.

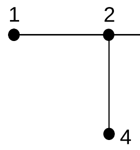
1.3 Non-Bipartite Graphs

In fact, Lovász gave an algorithm to detect perfect matchings in non-bipartite graphs. To describe this algorithm, we first need the notion of the *Tutte matrix* of a graph. Given an undirected graph $G = (V, E)$, its Tutte matrix T is the $n \times n$ symbolic matrix given by

$$T_{i,j} := \begin{cases} x_{i,j} & \text{if } i < j \text{ and } ij \in E \\ -x_{j,i} & \text{if } i > j \text{ and } ij \in E \\ 0 & \text{otherwise.} \end{cases}$$

Note that T is skew-symmetric.

Example 3 *As an example, consider the graph below.*



The Tutte matrix of this graph is given by

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

The rest of this subsection will be devoted to proving the following lemma. Once we have this lemma, a randomized algorithm for non-bipartite perfect matching follows from the Schwartz-Zippel lemma as in the previous subsection.

Lemma 3 *Let $G = (V, E)$ be an undirected graph and let T be its Tutte matrix. Then* $\det(T) \neq 0$ *if and only if G has a perfect matching.*

To prove this, we need to better understand the monomials that appear in the expansion of the determinant. Recall once more that the determinant of T is given by

$$\det(T) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} T_{i,\sigma(i)}.$$

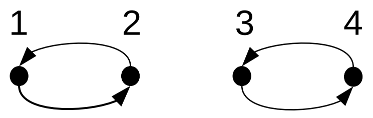
It will be useful to understand each permutation $\sigma \in S_n$ in terms of its cycle decomposition. For each $\sigma \in S_n$, we create a directed graph H_σ with vertices $\{1, \ldots, n\}$ and edges $\{(i, \sigma(i)) : 1 \le i \le n\}$. Note that if σ has a fixpoint, i.e., there is some i such that $\sigma(i) = i$, then H_σ has a self-loop.

By construction, every vertex in H_σ has in-degree 1 and out-degree 1. This implies that H_σ is a collection of disjoint directed cycles. Such a collection of edges is referred to as a *cycle cover*.[2] Remark that one can recover σ from H_σ.

Example 4 *For example, suppose that $\sigma \in S_4$ is the permutation given by*

$$\sigma(1) = 2 \qquad \sigma(2) = 1 \qquad \sigma(3) = 4 \qquad \sigma(4) = 3.$$

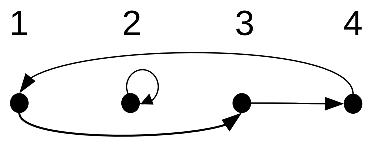
The corresponding graph H_σ is shown below.



As a second example, consider the permutation $\tau \in S_4$ given by

$$\tau(1) = 3 \qquad \tau(2) = 2 \qquad \tau(3) = 4 \qquad \tau(4) = 1.$$

The graph H_τ is shown below.



Note that the orientation of the graph is important.

We now prove a series of claims that relate the structure of a permutation σ and the corresponding term $\operatorname{sgn}(\sigma) \prod_{i=1}^{n} T_{i,\sigma(i)}$ in the expansion of $\det(T)$.

Claim 4 *Let $\sigma \in S_n$. Suppose σ has a fixpoint, or equivalently, that H_σ has a self-loop. Then $\prod_{i=1}^{n} T_{i,\sigma(i)} = 0$.*

Proof: Let j be a fixpoint of σ. By definition, $T_{j,j} = 0$, so the product $\prod_{i=1}^{n} T_{i,\sigma(i)}$ must also be zero. □

Claim 5 *Let $\sigma \in S_n$. Then $\prod_{i=1}^{n} T_{i,\sigma(i)} \ne 0$ if and only if for all $i \in \{1, \ldots, n\}$, the edge $\{i, \sigma(i)\}$ is in $E(G)$.*

[2]The term "cycle cover" may be somewhat misleading at first, as the cycles are required to be disjoint.

Proof: This follows from the definition of T. $\square$

Claim 6 *Let $\sigma \in S_n$. If H_σ consists entirely of even cycles and $\prod_{i=1}^n T_{i,\sigma(i)} \neq 0$, then G contains a perfect matching.*

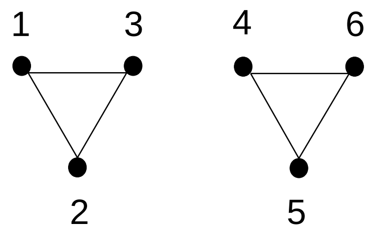
Proof: Since H_σ consists of even cycles, one can easily construct a perfect matching in H_σ. As $\prod_{i=1}^n T_{i,\sigma(i)} \neq 0$, the previous lemma implies that that the undirected versions of the edges in H_σ appear in G. Thus, the perfect matching in H_σ also appears in G, so G has a perfect matching. $\square$

The previous claim shows that if there is a permutation consisting entirely of even cycles that contributes a non-zero term to $\det(T)$, then G has a perfect matching. However, this is not enough to deduce an algorithm. We need to show that permutations which do not correspond to perfect matchings cancel out one another in the expression for the determinant.

Claim 7 *Let $\sigma \in S_n$. Let τ be the permutation obtained by reversing the direction of the arcs in C. Then $\operatorname{sgn}(\sigma) = \operatorname{sgn}(\tau)$.*

Proof: One can prove this via an alternative characterization of $\operatorname{sgn}(\sigma)$: the sign of a permutation is -1 if and only if σ consists of an odd number of even-length cycles. Reversing the direction of a cycle in σ clearly does not change the number of even-length cycles in σ. $\square$

To see why we will care about this, consider the following graph.



It is clear that this graph does not contain a perfect matching. However, the permutation $\sigma = (1,2,3)(4,5,6)$ contributes a non-zero term to $\det(T)$. Let $\tau = (3,2,1)(4,5,6)$ be the permutation obtained from σ by reversing the cycle $(1,2,3)$. We can use the previous claim along with the fact that T is skew-symmetric to show

$$\operatorname{sgn}(\sigma) \prod_{i=1}^n T_{i,\sigma(i)} = -\operatorname{sgn}(\tau) \prod_{i=1}^n T_{i,\tau(i)}.$$

That is, even though σ and τ contribute non-zero terms to $\det(T)$, they end up canceling one another out. We formalize this with the following claim.

Claim 8 *Let $S' = \{\sigma \in S_n \mid H_\sigma$ has a self-loop or an odd-length cycle$\}$. Then*

$$\sum_{\sigma \in S'} \operatorname{sgn}(\sigma) \prod_{i=1}^n T_{i,\sigma(i)} = 0.$$

Proof: Let $\sigma \in S'$. If H_σ has a self-loop, then by Claim **??**, we have $\operatorname{sgn}(\sigma) \prod_{i=1}^n T_{i,\sigma(i)} = 0$. If instead H_σ has no self-loops but has an odd-length cycle, let τ be the permutation obtained by reversing the odd-length cycle of σ which contains the lowest-numbered vertex.[3] It is easy to see that if we reverse the odd cycle in τ containing the lowest-numbered vertex, we obtain σ, so this procedure yields a matching between permutations. Then using Claim **??** and the fact that T is skew-symmetric, we have

$$\operatorname{sgn}(\sigma)\prod_{i=1}^{n} T_{i,\sigma(i)} = -\operatorname{sgn}(\tau)\prod_{i=1}^{n} T_{i,\tau(i)}.$$

This implies the terms corresponding to σ and τ cancel out one another. Hence

$$\sum_{\sigma\in S'} \operatorname{sgn}(\sigma)\prod_{i=1}^{n} T_{i,\sigma(i)} = 0$$

as claimed. □

We now show that terms corresponding to permutations outside of S' do not cancel one another out.

Claim 9 *Let S' be as in the previous claim and let $\sigma \in S_n \setminus S'$. If $\prod_{i=1}^{n} T_{i,\sigma(i)} \neq 0$, then $\det(T) \neq 0$*

Proof: By definition, σ has no self-loops and no odd cycles. Ignoring the sign for a moment, the monomial $\prod_{i=1}^{n} T_{i,\sigma(i)}$ consists of n variables that form a cycle cover of G. The only other permutations that produce this same monomial are those obtained by reversing the direction of a cycle in σ. As every cycle is even, reversing the direction of a cycle does not change the sign of the monomial $\prod_{i=1}^{n} T_{i,\sigma(i)}$. Moreover, Claim **??** implies that reversing a cycle does not change the sign of the corresponding permutation, so the non-zero monomial $\prod_{i=1}^{n} T_{i,\sigma(i)}$ never cancels out. □

We are now ready to prove that G contains a perfect matching if and only if $\det(T) \neq 0$. If G contains a perfect matching M, we can obtain from M a permutation σ consisting of length-two cycles. The previous claim implies that $\prod_{i=1}^{n} T_{i,\sigma(i)} \neq 0$, so we get that $\det(T) \neq 0$. Conversely, if $\det(T) \neq 0$, then there is some $\sigma \in S_n$ such that $\prod_{i=1}^{n} T_{i,\sigma(i)} \neq 0$ and σ consists entirely of even cycles. But then G contains a cycle cover consisting of even cycles, so G contains a perfect matching.

As previously remarked, we can now apply fast algorithms for computing the determinant and the Schwartz-Zippel lemma to obtain a randomized algorithm for non-bipartite perfect matching.

Theorem 10 *There is a randomized algorithm that in $O(n^{\omega})$ time checks whether an undirected graph G contains a perfect matching.*

Note that this algorithm has one-sided error. That is, if G does not contain a perfect matching, then the algorithm always reports this. If G contains a perfect matching, the algorithm correctly reports this with probability at least 1/2.

The following is left as an exercise to the reader.

Exercise 1 *Reduce the problem of checking whether G has a matching of size at least k to the problem of checking whether G has a perfect matching.*

[3]It doesn't matter exactly which cycle we reverse; it only matters that we make some concrete choice so that we end up with a matching on permutations with odd-length cycles.

1.4 Reducing Search to Decision

We now have a randomized algorithm to check whether a general graph G has a perfect matching. This raises a natural question: if G has a perfect matching, can we use this algorithm to find such a matching? A straightforward self-reduction will show that solving the decision problem is enough to solve the search problem.

Let $G = (V, E)$ be an undirected graph and let $e \in E$ be an edge with endpoints u and v. We can check if G has a perfect matching which contains e by checking whether the graph $G' := G - u - v$ has a perfect matching. If G' has a perfect matching M', then $M' + e$ is a perfect matching in G. Otherwise, we know that G does not have a perfect matching which contains e, so we recursively compute a perfect matching in the graph $G - e$.

Using the algorithm from the previous subsection, we can check if an n-vertex graph has a perfect matching in time $O(n^\omega)$, where ω is the exponent of matrix multiplication. We perform at most $m = |E|$ of these checks, so we obtain an algorithm with running time $O(mn^\omega)$. Using some clever tricks that are outside the scope of this lecture, one can shave off the $O(m)$ factor to obtain an algorithm with running time $O(n^\omega)$.

1.5 Parallel Algorithms for Matching

Given that we already have fast, deterministic algorithms for matching problems, do we gain anything using the linear algebraic approach? This approach yields algorithms which are randomized and whose running times improve over combinatorial algorithms only for dense graphs. Moreover, current algorithms for fast matrix multiplication are impractical to run on real-world data.

The main advantage of this algebraic viewpoint is that we can obtain fast *parallel* algorithms. It is currently an open problem to obtain a fast parallel combinatorial algorithm for matching.

We assume some familiarity with models of parallel computation, such as the PRAM model. In the PRAM model, we have a collection of processors which have access to shared memory and compute in lockstep. That is, at each step in time, every processor may read a location in memory, perform a step of computation, and possibly write a location in memory. The input to the problem is written in memory at the start of the computation.

We measure parallel time by the *depth* of the computation, which is the number of parallel steps taken by the processors. In this model, we consider a computation efficient if on an input of size n, the computation can be carried out in $\mathbf{poly}(\log(n))$ time on $\mathbf{poly}(n)$ processors. The class of problems admitting an efficient parallel algorithm is denoted by NC, short for "Nick's Class" and named after Nick Pippenger. If the depth of the computation can be bounded by $O(\log^i n)$ for some integer i, then the corresponding class is denoted NC^i. If we allow our algorithms to be randomized and have failure probability bounded by $1/3$, we obtain the classes RNC and RNC^i.

One has to be careful in modeling access to shared memory. Depending on whether processors are allowed to concurrently read or write the same memory location, one obtains different variants of the PRAM model.

- *Exclusive Read Exclusive Write (EREW)*: no two processors are allowed to simultaneously read from or write to the same memory location.
- *Exclusive Read Concurrent Write (ERCW)*: no two processors are allowed to read from the same memory location at the same time, but are allowed to write to the same memory location at the same time.

- *Concurrent Read Exclusive Write (CREW)*: all the processors are allowed to read from the same memory location at the same time, but only one processor may write a memory location at a time.
- *Concurrent Read Concurrent Write (CRCW)*: all the processors are allowed to read from or write to the same memory location at the same time.

The following is a major open problem.

Problem 1 *Is perfect matching in* NC*?*

Using determinants, one can obtain an RNC^2 algorithm for the decision version of perfect matching. To see this, recall that we can reduce computing determinants to matrix multiplication, which is easily solved in parallel via a standard divide-and-conquer strategy.

However, obtaining a randomized parallel algorithm to compute a perfect matching is not so straightforward. The search-to-decision reduction we gave in the previous subsection is sequential, and it is not clear how to implement a similar reduction in parallel. A 1987 paper of Mulmuley, Vazirani, and Vazirani gave an elegant solution to this problem. Their solution relied on an important observation known as the isolation lemma, which we now state.

Lemma 11 *Let $N = \{1, 2, \ldots, n\}$ be a finite ground set and let $\mathcal{F} \subseteq 2^N$ be a non-empty family of subsets. Suppose that for each $i \in N$ we pick weights w_i independently and uniformly at random from $\{1, 2, \ldots, d\}$. Then*

$$\Pr[\textit{there is a unique set } S \textit{ of minimum weight in } \mathcal{F}] \geq 1 - \frac{n}{d}.$$

In particular, if $d = 2n$, then there is a unique minimum-weight set in $\mathcal{F}$ with probability at least 1/2.

Note that the isolation lemma does not impose any structure on the family $\mathcal{F}$. The sets in $\mathcal{F}$ could correspond to matchings, spanning trees, or any other structure of interest.

Proof: Let $\mathcal{F} = \{S_1, \ldots, S_k\}$ for some $k \geq 1$. Write $w(S_j) := \sum_{i \in S_j} w_i$ for the weight of S_j. For $i \in N$, define

$$\alpha_i := \min_{S_j : i \notin S_j} w(S_j) - \min_{S_j : i \in S_j} (w(S_j) - w_i).$$

If $\alpha_i = w_i$, we say that i is *ambiguous.* We will show that if the minimum-weight set is not unique, then there is an ambiguous element. We then bound the probability of there being an ambiguous element.

Suppose the minimum-weight set is not unique. Without loss of generality, suppose that S_1 and S_2 are both sets of minimum weight and that there is some $i \in S_1 \setminus S_2$. We will show that i is ambiguous. By assumption, S_2 is a minimum-weight set among sets not containing i and S_1 is a minimum-weight set among sets containing i. Since S_1 and S_2 are both minimum-weight sets, we have $w(S_1) = w(S_2)$. This gives us

$$\begin{aligned}\alpha_i &= w(S_2) - (w(S_1) - w_i)\\ &= w(S_1) - w(S_1) + w_i\\ &= w_i.\end{aligned}$$

Hence there is an ambiguous element, namely i.

Now we bound the probability that there is an ambiguous element. Consider the probability that i is ambiguous for some fixed $i \in N$. If we fix the weights of $N \setminus \{i\}$, then we have fully determined both $\min_{S_j : i \notin S_j} w(S_j)$ and $\min_{S_j : i \in S_j}(w(S_j) - w_i)$. Thus, we know the exact value of α_i before w_i is chosen. There is a $1/d$ chance that w_i is chosen to be α_i and hence a $1/d$ chance that i is ambiguous. Since this holds regardless of the choice of weights for $N \setminus \{i\}$, we get that i is ambiguous with probability exactly $1/d$. By a union bound, we have

$$\begin{aligned}\Pr[\exists i \text{ such that } i \text{ is ambiguous}] &\leq \sum_{i \in N} \Pr[i \text{ is ambiguous}] \\ &\leq \frac{n}{d}.\end{aligned}$$

Thus, with probability at least $1 - \frac{n}{d}$, there is no ambiguous element, and hence there is a unique minimum-weight set with probability at least $1 - \frac{n}{d}$. □

With the isolation lemma in hand, we can describe the approach of Mulmuley, Vazirani, and Vazirani to design a randomized parallel algorithm to find a perfect matching. We start with the following claim.

Claim 12 *Let $G = (V, E)$ be an undirected graph with weights $w : E \to \mathbb{R}_{\geq 0}$. Suppose that there is a* unique *minimum-weight perfect matching M in G of weight W. Let $e \in E$ be an edge with endpoints u and v. Let W_e be the weight of a minimum-weight perfect matching in the graph $G - u - v$. Then $e \in M$ if and only if $w(e) + W_e = W$.*

Proof: Let M' be the min-weight perfect matching in $G - u - v$. Since $M' + e$ is a perfect matching in G, we must have $w(e) + W_e \geq W$.

If $w(e) + W_e = W$, then $M' + e$ is a minimum-weight perfect matching in G. Since this matching is unique, we have $e \in M$.

Conversely, suppose $e \in M$. Then $M - e$ is a perfect matching in G' of weight $W - w(e)$. Since M' is a minimum-weight perfect matching in G', we have $W - w(e) \geq W_e$. Putting toegether these inequalities gives

$$W \leq w(e) + W_e \leq W,$$

so $W = w(e) + W_e$. □

So far, we've only seen how to test if a graph has a perfect matching by evaluating the determinant of the Tutte matrix at a random point. This does not immediately give us an algorithm to find a minimum-weight matching, or even compute the weight of a minimum-weight matching. However, we can combine the isolation lemma along with the Tutte matrix to give an algorithm that can compute the weight of a minimum-weight perfect matching, provided that the input graph has a unique perfect matching of minimum weight.

Lemma 13 *Let $G = (V, E)$ be an undirected graph with edge weights $w : E \to \mathbb{N}$. Suppose that G has a unique minimum-weight perfect matching of weight W. Let T be the evaluation of the Tutte matrix under the substitution $x_{i,j} \mapsto 2^{w(ij)}$. Then 2^{2W} is the highest power of 2 that divides $|\det(T)|$.*

Proof: Recall once more the expansion of the determinant

$$\det(T) = \sum_{\sigma \in S_n} \operatorname{sgn}(\sigma) \prod_{i=1}^{n} T_{i,\sigma(i)},$$

where

$$T_{i,j} = \begin{cases} 2^{w(ij)} & i < j \text{ and } ij \in E \\ -2^{w(ji)} & i > j \text{ and } ij \in E \\ 0 & \text{otherwise.} \end{cases}$$

Recall that terms corresponding to permutations with odd cycles cancel out. For the permutation $\sigma \in S_n$ corresponding to the unique minimum-weight perfect matching[4] in G, we have $\prod_{i=1}^{n} T_{i,\sigma(i)} = \pm 2^{2W}$. We will show that for all other permutations $\tau \in S_n$, we have $\prod_{i=1}^{n} T_{i,\tau(i)} = \pm 2^{2a}$ for some integer $a > W$, which completes the proof.

Let $\tau \in S_n$ be a permutation not corresponding to the unique minimum-weight perfect matching in G. If τ corresponds to a matching of weight a, then $\prod_{i=1}^{n} T_{i,\tau(i)} = \pm 2^{2a}$. Since this matching is not the one of minimum-weight, we have $a > w$ as required.

Suppose instead that $\tau \in S_n$ is a permutation consisting solely of even cycles, but not necessarily corresponding to a matching in G. With some care, one can show that in this case, we still have $\prod_{i=1}^{n} T_{i,\tau(i)} = \pm 2^{2a}$ for some $a > W$. This amounts to checking that the sum of weights on an even cycle cannot be smaller than twice the weight of a minimum-weight perfect matching contained in that cycle. We leave the details to the interested reader. $\square$

We are finally ready to describe the algorithm of Mulmuley, Vazirani, and Vazirani. Given an undirected graph $G = (V, E)$, we do the following.

1. Independently for each edge $e \in E$, pick a weight w_e uniformly at random from $\{1, 2, \ldots, 2 \cdot |E|\}$.

2. Compute the determinant of the Tutte matrix T, $|\det(T_G)|$, under the evaluation $x_{i,j} \mapsto 2^{w_{ij}}$. Let 2^{2W} be the highest power of 2 that divides $|\det(T)|$.

3. For each $e \in E$ with endpoints u and v in parallel, compute $\det(T_{-e})$ where T_{-e} is the matrix obtained from T by removing the columns and rows corresponding to u and v. If $\det(T_{-e}) \neq 0$, let 2^{2a} be the highest power of 2 that divides $|\det(T_{-e})|$. If $a + w_e = W$, then add e to M.

4. Check that M is a perfect matching. If yes, output M. Otherwise, report that there is no perfect matching in G.

One can put together the previous claims and lemmas to show that this algorithm succeeds with probability at least 1/2. There are some subtle issues in that the numbers involved in this algorithm are very large, possibly of bit-size $O(n^2)$. This can be remedied, as standard arithmetic operations can be implemented in parallel in $O(\log n)$ time. Overall, this yields a randomized parallel algorithm which uses $\mathbf{poly}(n)$ work and $O(\log^2 n)$ depth.

Recent breakthroughs have established *deterministic* algorithms for perfect matching which use $2^{\mathbf{poly}(\log n)}$ work but $\mathbf{poly}(\log n)$ depth. See the March 2019 CACM article by Fenner, Gurjar, and Thierauf for more on this.

[4]If i and j are matched in the minimum-weight perfect matching, then $\sigma(i) = j$ and $\sigma(j) = i$.