2.4 Lecture 11 (26/Oct): Perfect matchings in general graphs. Parallel computation. Isolating lemma.

2.4.1 Deciding existence of a perfect matching in a graph

Bipartite graphs were handled last time. Now we consider general graphs.

Deterministically, deciding the existence of a perfect matching in a general graph is harder than the same problem in a bipartite graph. (As noted, we have poly-time algorithms, but not nearly so simple ones.) With randomization, however, we can adapt the same approach to work with almost equal efficiency.

We must define the Tutte matrix of a graph $G = (V, E)$. Order the vertices arbitrarily from $1, \ldots, n$ and set

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

Theorem 38 (Tutte [104]) $\mathrm{Det}(T) \neq 0$ *iff T has a perfect matching.*

This determinant is not multilinear in the variables, so the lemma from last time does not apply.

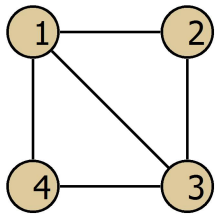
Proof: If T has a perfect matching, assign $x_{ij} = 1$ for edges in the matching, and 0 otherwise. Each matching edge $\{i, j\}$ describes a transposition of the vertices i, j. With this assignment every row and column has a single nonzero entry corresponding to the matching edge it is part of, so the matrix is the permutation matrix (with some signs) of the involution that transposes the vertices on each edge. Since a transposition has sign -1 and there is a single -1 in each pair of nonzero entries, the contribution of each transposition to the determinant is 1, and overall we have $\mathrm{Det}(T) = 1$.

Conversely suppose $\mathrm{Det}(T) \neq 0$ as a polynomial. Consider the determinant as a signed sum over permutations. The net contribution to the determinant from all permutations having an odd cycle is 0, for the following reason. In each such cycle identify the "least" odd cycle by some criterion, e.g., ordering the cycles by their least-indexed vertex. Then flip the direction of the least odd cycle. This map is an involution on the set of permutations. It carries the permutation to another, which contributes the opposite sign to the determinant, since the sign of all edges in the cycle flipped. (Figure 2.2.)

$$\left(\begin{array}{cc|ccc} & \cdots & \cdots & \cdots & \cdots \\ \cdots & & \cdots & \cdots & \cdots \\ \hline \cdots & \cdots & & x_{34} & \cdots \\ \cdots & \cdots & \cdots & & x_{45} \\ \cdots & \cdots & -x_{35} & \cdots & \end{array}\right) \quad \text{vs.} \quad \left(\begin{array}{cc|ccc} & \cdots & \cdots & \cdots & \cdots \\ \cdots & & \cdots & \cdots & \cdots \\ \hline \cdots & \cdots & & \cdots & x_{35} \\ \cdots & \cdots & -x_{34} & & \cdots \\ \cdots & \cdots & \cdots & -x_{45} & \end{array}\right) \tag{2.2}$$

Figure 2.2: Flipping a cycle among vertices $3, 4, 5$. Preserves permutation sign; reverses signs of cycle variables.

Therefore there are permutations of the vertices, supported by T (i.e., each vertex is mapped to a destination along one of the edges incident to it, that is, $\pi(i) = j \Rightarrow T_{ij} \neq 0$), having only even cycles. The even cycles of length 2 are matching edges, and in any even cycle of length greater than 2, we can use every alternate edge; altogether we obtain a perfect matching. See Figure 2.3 for a graph having perfect matchings, and two permutations from which we can read off perfect matchings. □



$$\begin{pmatrix} & x_{12} & x_{13} & x_{14} \\ -x_{12} & & x_{23} & \\ -x_{13} & -x_{23} & & x_{34} \\ -x_{14} & & -x_{34} & \end{pmatrix} \tag{2.3}$$

$$\begin{pmatrix} & * & & \\ * & & & \\ & & & * \\ & & * & \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} & * & & \\ & & * & \\ & & & * \\ * & & & \end{pmatrix} \tag{2.4}$$

Figure 2.3: A graph and its Tutte matrix with two different permutations π from which we can read off perfect matchings: the involution $(12)(34)$ and the 4-cycle (1234).

In exactly the same way as for the bipartite case, this yields:

Theorem 39 *The algorithm to determine existence of a perfect matching in a graph on n vertices, is error-free on graphs lacking a perfect matching, and the probability of error of the algorithm on graphs which have a perfect matching, is at most n/q. The runtime of the algorithm is $n^{\omega+o(1)}$, where ω is the matrix multiplication exponent.*

By self-reducibility this immediately yields an $\tilde{O}(n^{\omega+2})$-time algorithm for *finding* a perfect matching. (Remove an edge, see if there is still a perfect matching without it,...)

2.4.2 Parallel computation

There are two major processes at work in the above algorithm: determinant computations, and sequential branching used in the self-reducibility argument. As we discuss in a moment, the linear algebra can be parallelized. But the branching is inherently sequential.

Nevertheless, we will shortly see a completely different algorithm, that avoids this sequential branching. In this way we'll put the problem of finding a perfect matching, in RNC.

NC^i = problems solvable deterministically by $\text{poly}(n)$ processors in $\log^i n$ time. (Equivalently, by $\text{poly}(n)$ -size, $\log^i n$ -depth circuits.)

$NC = \bigcup_i NC^i$

RNC^i, RNC = same but the processors (gates) may use random bits.

(Note, we are glossing over the "uniformity" conditions of the complexity classes.)

2.4.3 Sequential and Parallel Linear Algebra

In sequential computation, there are reductions showing that matrix inversion and multiplication have the same time complexity (up to a factor of $\log n$), and that determinant is no harder than these.

In parallel computation, the picture is actually a little simpler. Matrix multiplication is in NC^1 (right from the product definition, since we can use a tree of depth $\log n$ to sum the n terms of a row-column inner product). Matrix inversion and determinant are in NC^2, due to Csanky [28] (over fields of characteristic 0) (and using fewer processors in RNC^2 by Pan [88]); the problem is also in NC over general fields [15, 23].

Csanky's algorithm builds on the result of Valiant, Skyum, Berkowitz and Rackoff [105] that any deterministic sequential algorithm computing a polynomial of degree d in time m can be converted to a deterministic parallel algorithm computing the same polynomial in time $O((\log d)(\log d + \log m))$ using $O(d^6 m^3)$ processors.

For a good explanation of Csanky's algorithm see [67] §31, and for more on parallel algorithms see [72].

2.4.4 *Finding* Perfect Matchings in General Graphs. The Isolating Lemma

We now develop a randomized method of Mulmuley, U. Vazirani and V. Vazirani [82] to find a perfect matching if one exists. A polynomial time algorithm is implied by the previous testing method along with self-reducibility of the perfect matching decision problem. However, with the following method we can solve the same problem in *parallel*, that is to say, in polylog depth on polynomially many processors. This is not actually the first RNC algorithm for this task—that is an RNC^3 method due to [62]—but it is the "most parallel" since it solves the problem in RNC^2.

A slight variant of the method yields a *minimum weight perfect matching* in a weighted graph that has "small" weights, that is, integer weights represented in unary; and there is a fairly standard reduction from the problem of finding a *maximum matching* to finding a minimum weight perfect matching in a graph with weights in $\{0, 1\}$. So we actually through this method can find a maximum matching in a general graph, with a similar total amount of work.

There are really two key ingredients to this algorithm. The first, which we have already noted, is that all basic linear algebra problems can be solved in NC^2.

The second ingredient, which will be our focus, is the following lemma. First some notation. Let $A = \{a_1, a_2, \ldots, a_m\}$ be a finite set. If $a_1, \ldots, a_m$ are assigned weights $w_1, \ldots, w_m$, the weight of set $S \subseteq A$ is defined to be $w(S) = \sum_{a_i \in S} w_i$. Let $\mathcal{S} = \{S_1, \ldots, S_k\}$ be a collection of subsets of A. Let $\min(\mathcal{S} : w) \subseteq \mathcal{S}$ be the collection of those S of least weight in $\mathcal{S}$. We are interested in the event that the least weight is uniquely attained, i.e., the event that $|\min(\mathcal{S} : w)| = 1$.

Lemma 40 ([82] Isolating Lemma, based on improved version in [100]) *Let the weights $w_1, \ldots, w_m$ be independent random variables, each w_i being sampled uniformly in a set $R_i \subseteq \mathbb{R}$, $|R_i| \geq r \geq 2$. Then*

$$\Pr(|\min(\mathcal{S} : w)| = 1) \geq (1 - 1/r)^m. \tag{2.5}$$

This lemma is remarkable because of the absence of a dependence on k, the size of the family, in the conclusion.