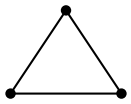
When we first studied network flows, we saw how to find maximum cardinality matchings in bipartite graphs — just add in a super-source s to the nodes on the left and a super-sink t to the nodes on the right, direct all edges from left to right and put unit capacity on them, and find a max-flow. Simple and effective.

But what if the graph is not bipartite? The reduction fails. E.g., consider the problem of finding a maximum cardinality matching on the 3-cycle.



Not clear any more how to just reduce this to finding a max-flow.

Today we'll see one way to find maximum cardinality matchings in general graphs, based on polynomials and randomization. Another approach that is purely graph-theoretic is the Blossoms Algorithm of Jack Edmonds; check out Danny Sleator's notes from Fall 2012 version of the course.

1 Reductions from Maximum Matchings to Perfect Matchings

We can reduce maximum cardinality matching to finding perfect matchings. Formally, if we can find perfect matchings (or correctly say that there is no perfect matching) in any graph on n vertices in time $T(n)$ then we can find a max-cardinality matching in time $O(T(n) \log n)$.

How? Suppose we wanted to check if there was a matching with at least K edges in G. If there were, $2K$ nodes would be matched, and $n - 2K$ would be free. So add $n - 2K$ new nodes to G, and add edges between these new nodes and all old nodes in G. This new graph, H, has a perfect matching if and only if G has a matching with at least K edges. (Exercise: prove this.) Now you can binary search on the value of K.

A similar reduction holds while preserving bipartiteness. (Exercise: how does this reduction go?) I.e., if we can find perfect matchings (or correctly say that there is no perfect matching) in any bipartite graph on n vertices in time $T(n)$ then we can find a maximum-cardinality matching in any bipartite graph in time $O(T(n) \log n)$.

So, from now on, let's just focus the perfect matching problem.

2 An Algebraic Approach to Perfect Matchings

This idea is really out of left field. Let's begin easy and first try to solve the perfect matching problem on bipartite graphs, which we know how to solve using flows. We'll get a very different algorithm that will generalize to the non-bipartite case. In fact, let's just solve the *decision version* of the problem: *given a bipartite graph $G = (U, V, E)$, does there exist a perfect matching in G?* A simple yes/no answer will suffice for now. We will assume that $|U| = |V| = n$, else the answer is clearly no.

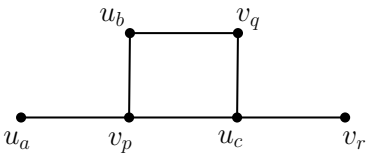
2.1 The Tutte Matrix

This approach will use the following object:[1]

Definition 1 *The* Tutte matrix[2] *of bipartite graph $G = (U, V, E)$ is an $|U| \times |V|$ matrix $M_b(G)$ with the entry at row i and column j,*

$$M_b(G)_{i,j} = \begin{cases} 0 & \text{if } (u_i, v_j) \notin E \\ x_{i,j} & \text{if } (u_i, v_j) \in E \end{cases}$$

This is a matrix of variables $x_{i,j}$. It is square, since we assumed that $|U| = |V|$. E.g., for this graph (which does have a perfect matching)



the matrix looks like

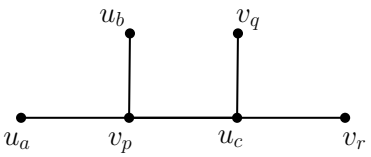
$$M_b(G) = \begin{pmatrix} x_{a,p} & 0 & 0 \\ x_{b,p} & x_{b,q} & 0 \\ x_{c,p} & x_{c,q} & x_{c,r} \end{pmatrix} \tag{1}$$

Just a square matrix with some variables in it. And just like any matrix, we can take its determinant. For example, the determinant in the example would be

$$\begin{aligned} \det(M_b(G)) &= \begin{vmatrix} x_{a,p} & 0 & 0 \\ x_{b,p} & x_{b,q} & 0 \\ x_{c,p} & x_{c,q} & x_{c,r} \end{vmatrix} \\ &= x_{a,p}((x_{b,q}x_{c,r}) - (x_{c,q} \cdot 0)) - 0(\cdots) + 0(\cdots) \\ &= x_{a,p}x_{b,q}x_{c,r}. \end{aligned}$$

The determinant is a (multivariate) polynomial in the variables $\{x_{i,j}\}_{\{i,j\} \in E}$. Usually your matrix has numbers in it, and the determinant is thus a number. Here we are thinking of the entries of the matrix as being variables, so the determinant is a polynomial.

Let's do another example:



[1] The Tutte matrix is named after the late Bill Tutte, a great graph theorist and one of the founders of modern graph theory. He was code-breaker in World War II, and here is a nice account of his work at Bletchley Park.

[2] Strictly speaking, in 1947 Tutte came up the matrix for general graphs you'll see later in this lecture, and this one for bipartite graphs was first formally defined by Jack Edmonds (1967), but we'll stick with calling it the Tutte matrix as well.

The Tutte matrix $M_b(G)$ now changes, and its determinant is

$$\begin{aligned}\det(M_b(G)) &= \begin{vmatrix} x_{a,p} & 0 & 0 \\ x_{b,p} & 0 & 0 \\ x_{c,p} & x_{c,q} & x_{c,r} \end{vmatrix} \\ &= x_{a,p}((0 \cdot x_{c,r}) - (x_{c,q} \cdot 0)) - 0(\cdots) + 0(\cdots) \\ &= 0.\end{aligned}$$

This time the determinant is the polynomial 0. By the way, does this graph have a perfect matching? (No.)

We denote the determinant of the Tutte matrix $M_b(G)$ by $P_G(\mathbf{x})$, where $\mathbf{x}$ is the vector of all the $x_{i,j}$ variables. The main theorem we will need is the following:

Theorem 2 *A bipartite graph G has a perfect matching if and only if $P_G(\mathbf{x})$, the determinant of the Tutte matrix, is not the zero polynomial.*

Proof: We have the following expression for the determinant :

$$\det(M) = \sum_{\pi \in S_n} (-1)^{sgn(\pi)} \prod_{i=1}^{n} M_{i,\pi(i)}$$

where S_n is the set of all permutations on $[n]$, and $sgn(\pi)$ is the sign of the permutation π. There is a one to one correspondence between a permutation $\pi \in S_n$ and a (possible) perfect matching $\{(u_1, v_{\pi(1)}), (u_2, v_{\pi(2)}), \cdots, (u_n, v_{\pi(n)})\}$ in G. Note that if this perfect matching does not exist in G (*i.e.* some edge $(u_i, v_{\pi(i)}) \notin E$) then the term corresponding to π in the summation is 0. So we have

$$\det(M) = \sum_{\pi \in P} (-1)^{sgn(\pi)} \prod_{i=1}^{n} x_{i,\pi(i)}$$

where P is the set of perfect matchings in G. This is clearly zero if $P = \emptyset$, *i.e.*, if G has no perfect matching. If G has a perfect matching, there is a $\pi \in P$ and the term corresponding to π is $\prod_{i=1}^{n} x_{i,\pi(i)} \neq 0$. Additionally, there is no other term in the summation that contains the same set of variables. Therefore, this term is not cancelled by any other term. So in this case, $\det(M) \neq 0$. ■

2.2 Using the Tutte Matrix to Decide Existence of Perfect Matchings

Great! So we now have a test for whether G has a perfect matching—all we have to do is check if the Tutte matrix's determinant $P_G(\mathbf{x}) = \det(M_b(G))$ is equal to the zero polynomial. And the determinant can't be that hard to compute, right?

Well...

If the matrix was composed of numbers, it would not be hard to compute the determinant.[3] But we have a matrix full of variables. The polynomial you get from the determinant could be a pain to compute, with many many terms, a lot of cancellations that lead to the zero polynomial (or not). For example, if we took the following

Thankfully, we can get around this. We will soon prove the following theorem in Section 3.

[3]How? The definition of the determinant is in terms of a sum over all $n!$ permutations. This is the first (and often only) algorithm we see to evaluate the determinant, but it is a very inefficient algorithm, taking $n!$ time. How does one do better? One simple case is when the matrix is lower-triangular, say, then the determinant is just the product of the diagonal values. And you can use Gaussian elimination to convert any matrix into an upper triangular matrix without changing the determinant value, see, e.g., the Wikipedia entry. This takes $O(n^3)$ math operations.

Theorem 3 *There is a randomized polynomial-time algorithm $\mathcal{A}$ that, given black-box (evaluation) access to a N-variable degree-d polynomial $P(\mathbf{x})$, outputs a correct answer to the question "Is $P(\mathbf{x})$ the zero polynomial?" with probability at least $1/2$.*

Note that this algorithm does not require $P(\mathbf{x})$ to be given explicitly. It just assumes we have some way to evaluate $P()$ at any given point $\mathbf{v}$ — this is the "black-box (evaluation) access" part. We assume that each call to this black-box takes unit time. Of course, if actually evaluating $P()$ at some point $\mathbf{v}$ takes T time, we just multiply the running time by T—no sweat.

Using this theorem, we're done. We can just use this algorithm on the Tutte matrix's determinant. By Theorem 2 and Theorem 3, with probability at least $1/2$, the algorithm $\mathcal{A}$ will correctly tell us whether $P_G()$ is the zero polynomial or not, and hence whether the bipartite graph G has a perfect matching or not. And by repeating this test $3\log_2 n$ times indepedently, we can make the probability of error $2^{-3\log_2 n} = 1/n^3$. Tiny.

What's the running time? The running time would be polynomial, as long as we could evaluate the polynomial fast. And indeed, given some values $v_{i,j}$ for the $x_{i,j}$ variables, we can evaluate the determinant of the Tutte matrix at some set of numerical values $v_{i,j}$ by plugging in these values and computing the determinant in $O(n^3)$ time (using Gaussian elimination).

2.3 How About Finding the Matching?

So far we were just solving the decision version of the problem. How about the search version? (You want to find the perfect matching.) Well, you've seen such reductions before. Here's how to do it in this context:

1. Pick $(u_i, v_j) \in E$.
2. Check if $G \backslash \{u_i, v_j\}$ has a perfect matching (using the decision version of the problem).
3. If "Yes", output (u_i, v_j) to be in the matching and recurse on $G \backslash \{u_i, v_j\}$, the graph obtained after the removal of vertices u_i and v_j (and the edges incident to them).
4. If "No", recurse on $G - (u_i, v_j)$, the graph obtained after removing just the single edge (u_i, v_j).

Of course, our decision version made mistakes, with small probability $1/n^3$. So we need to make sure this algorithm does not make *any* mistakes with some reasonable probability. To do this, how many times do we execute Step 2? At most $|E| \leq n^2$ times. So the probability that the algorithm $\mathcal{A}$ made an error in any of these $|E|$ steps is at most $|E| \cdot 1/n^3 \leq 1/n$. And if $\mathcal{A}$ did not err, we will find a perfect matching at the end of the process.

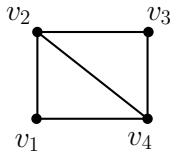
2.4 Non-Bipartite Matching

How do we extend this to general matchings? Easy. We just need a matrix $M_g(G)$ whose determinant is not the zero polynomial if and only if the graph G has a perfect matching.

And this is exactly what Tutte gave us, back in 1947. For any graph $G = (V, E)$ with vertices $v_1, v_2, \ldots, v_n$, he defined a $|V| \times |V|$ matrix $M_g(G)$ as follows:

$$M_g(G)_{i,j} = \begin{cases} x_{i,j} & \text{if } \{v_i, v_j\} \in E \text{ and } i < j \\ -x_{j,i} & \text{if } \{v_i, v_j\} \in E \text{ and } i > j \\ 0 & \text{if } (v_i, v_j) \notin E \end{cases}$$

Again, the associated polynomial for the (general) graph G is defined as $P_G(\mathbf{x}) := \det(M_g(G))$. For an example, consider



The Tutte matrix is now:

$$M_g(G) = \begin{pmatrix} 0 & x_{1,2} & 0 & x_{1,4} \\ -x_{1,2} & 0 & x_{2,3} & x_{2,4} \\ 0 & -x_{2,3} & 0 & x_{3,4} \\ -x_{1,4} & -x_{2,4} & -x_{3,4} & 0 \end{pmatrix} \tag{2}$$

and its determinant is:

$$P_G(\mathbf{x}) = \det(M_g(G)) = {x_{1,2}}^2{x_{3,4}}^2 + 2\,x_{1,2}x_{2,3}x_{3,4}x_{1,4} + {x_{2,3}}^2{x_{1,4}}^2 = (x_{1,2}x_{3,4} + x_{1,3}x_{2,4})^2$$

We won't prove this theorem here, but you might want to try to prove it for yourself.

Theorem 4 *A (general) graph G has a perfect matching if and only if $P_G(\mathbf{x})$, the determinant of the Tutte matrix, is not the zero polynomial.*

3 Polynomial Identity Checking

So now it remains to prove Theorem 3, to check if a polynomial (given as a black-box) is the zero polynomial or not. Let's formulate a more general-seeming problem (which is the same problem in disguise). In the *polynomial identity checking* problem, we are given two multi-variate polynomials $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$ each with degree d; again we are computing in some field $\mathbb{F}$.[4] We may not be given the polynomials explicity, so we may not be able to read the polynomials in poly-time — we just have "black-box" access for evaluating a polynomial. Given these two polynomials, the problems is to determine if the polynomials are equal: i.e., if $f = g$, or equivalently, $f - g = 0$. Letting $Q = f - g$, it suffices to check if a given polynomial is identically zero. There is no known poly-time algorithm for this problem. But we will now show that it has a randomized poly-time algorithm.

First consider the univariate case. We can pick $d+1$ distinct values at random from $\mathbb{F}$. If $Q(x) = 0$ for all $d + 1$ values for x, then $Q = 0$. This follows from the basic and super-useful fact, that for any field $\mathbb{F}$, a polynomial of degree at most d over that field can have at most d roots.

This approach does not directly apply to the multivariate case; in fact, the polynomial over two variables $f(x, y) = xy - 3$ over the reals has an *infinite number* of roots. Over the finite field $\mathbb{F}_q$, the degree-d polynomial over n variables

$$Q(x_1, x_2, \ldots, x_n) = (x_1 - 1)(x_1 - 2) \cdots (x_1 - d)$$

has dq^{n-1} roots (when $d \leq q = |\mathbb{F}|$).

However, things still work out. Roughly speaking, we can handle the multivariate case by fixing $n - 1$ variables and applying the result from the univariate case. Consider the following algorithm, which assumes we have some subset $S \subset \mathbb{F}$ with $|S| \geq 2d$.

- Pick $r_1, \ldots, r_n \in_R S$.

[4] You can imagine we are working over the field $\mathbb{R}$ of reals for now. In practice you may want over a smaller field, to avoid numerical issues.

- Evaluate $Q(r_1, \ldots, r_n)$.
- If 0, return $Q = 0$.

Theorem 5 (Schwartz (1980), Zippel (1979)) *If, in the above algorithm, the polynomial $Q \neq 0$, we have*

$$\Pr[Q(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}.$$

Proof: By induction on n. The base case is the univariate case described above. With $Q \neq 0$, we want to compute $\Pr[Q(r_1, \ldots, r_n) = 0]$. Let k be the largest power of x_1. We can rewrite

$$Q(x_1, \ldots, x_n) = x_1^k A(x_2, \ldots, x_n) + B(x_1, \ldots, x_n)$$

for some polynomials A and B. Now we consider two events. Let $\mathcal{E}_1$ be the event that $Q(r_1, \cdots, r_n)$ evaluates to 0, and $\mathcal{E}_2$ be the event that $A(r_2, \cdots, r_n)$ evaluates to 0.

We can rewrite the probability that $Q(r_2, \cdots, r_n)$ is 0 as:

$$\begin{aligned} \Pr[Q(r) = 0] = \Pr[\mathcal{E}_1] &= \Pr[\mathcal{E}_1 \mid \mathcal{E}_2] \Pr[\mathcal{E}_2] + \Pr[\mathcal{E}_1 \mid \neg\mathcal{E}_2] \Pr[\neg\mathcal{E}_2] \\ &\leq \Pr[\mathcal{E}_2] + \Pr[\mathcal{E}_1 \mid \neg\mathcal{E}_2] \end{aligned}$$

Let us first bound the probability of $\mathcal{E}_2$, or the probability that $A(r_2, \cdots, r_n) = 0$. The polynomial A has degree $d - k$ and fewer varaibles, so we can use the inductive hypothesis to obtain

$$\Pr[\mathcal{E}_2] = \Pr[A(r_2, \ldots, r_n) = 0] \leq \frac{d-k}{|S|}.$$

Similarly, given $\neg\mathcal{E}_2$ (or $A(r_2, \cdots, r_n) \neq 0$), the univariate polynomial $Q(x_1, r_2, \ldots, r_n)$ has degree k. Therefore, again by inductive hypothesis,

$$\Pr[\mathcal{E}_1 \mid \neg\mathcal{E}_2] = \Pr[Q(x_1, r_2, \ldots, r_n) = 0 \mid A(r_2, \ldots, r_n) \neq 0] \leq \frac{k}{|S|}.$$

We can substitute into the expression above to get

$$\begin{aligned} \Pr[Q(r) = 0] &\leq \Pr[\mathcal{E}_2] + \Pr[\mathcal{E}_1 \mid \neg\mathcal{E}_2] \\ &\leq \frac{d-k}{|S|} + \frac{k}{|S|} = \frac{d}{|S|} \end{aligned}$$

This completes the inductive step. ■

Polynomial identity testing is a powerful tool, both in algorithms and in complexity theory. We can use it to find matchings in parallel, and it arises all over the place. Also, as mentioned above, there is no poly-time deterministic algorithm currently known for this problem; this is one of the few problems for which the only efficient algorithm we know is randomized.