Maximum Matchings via the Tutte Matrix

Eli Ross
April 29, 2013
18.304: Seminar in Discrete Math
Introduction

- **Bipartite graph**: Two sets of vertices (we take both to have size $n$) such that there is no edge within the same set.
- **Matching**: A set of edges chosen in a graph such that no vertex is the endpoint of two or more edges.
- **Perfect Matching**: A matching which contains every vertex as an endpoint of some edge in the matching.
Problem

• We want to answer the question:

*Does an unweighted, undirected graph (bipartite or general) G admit a perfect matching? (If so, find one.)*

• We turn to matrices and randomized algorithms to help answer this question.
Outline

• Refreshing Old Topics (Lin. Alg, Permutations)
• A First Attempt (Naïve Algorithm)
• Schwartz-Zippel Lemma for Polynomial Equality
• Good Method for Bipartite Graphs
• Refinement for General Graphs
• Discussion and Comparison
Recall...

- \( \det(A) = \sum_{\pi \in S_n} \left( \text{sgn}(\pi) \cdot \prod_{i=1}^{n} a_{i\pi(i)} \right) \) where \( S_n \) is the set of all permutations of \( \{1,2,\ldots,n\} \) and

\[
\text{sgn}(\pi) = \begin{cases} 
1 & \text{if even number of transpositions} \\
-1 & \text{if odd number of transpositions}
\end{cases}
\]

- All permutations can be written in terms of cycles
- A bipartite matching is essentially a permutation
Preliminary Idea

• Define $A$ as follows:

$$a_{ij} = \begin{cases} 
1 & \text{if } i \sim j, \ i \in U, \ j \in V \\
0 & \text{otherwise}
\end{cases}$$

• **Claim:** If $\det(A) \neq 0$, then $G$ has a perfect matching.
Claim: If $\det(A) \neq 0$, then $G$ has a perfect matching.

- Recall that:

$$
\det(A) = \sum_{\pi \in S_n} \left( \text{sgn}(\pi) \cdot \prod_{i=1}^{n} a_{i\pi(i)} \right)
$$

- If determinant is non-zero, then there must be some non-zero $\prod_{i=1}^{n} a_{i\pi(i)}$ term, so a perfect matching exists. (Why?)

- A quick example might help you believe me!
How Good Is This?

• Computing the determinant of this matrix will take

\[ O(n^{2.376}) \]

via the Coppersmith-Winograd algorithm. (Pretty impractical, so actual runtime is probably worse.)

• However, what’s the big problem here?
A Better Matrix!

• Define $B$ as follows:

$$b_{ij} = \begin{cases} x_{ij} & \text{if } i \sim j, \ i \in U, \ j \in V \\ 0 & \text{otherwise} \end{cases}$$

• **Claim:** $\det(B) \neq 0$ if and only if $G$ has a perfect matching. (We’ll come back to this.)
Testing Polynomial Equality

• Question: does $f(x) = g(x)$? How could we tell without actually finding the polynomials?
• Idea: substitute random values from

$$\{1, 2, \ldots, mn\}$$

and see if the result is 0, where

$$\deg(f(x)), \deg(g(x)) \leq n.$$
Probability of Failure

• The most roots of \( f(x) - g(x) \) in \( \{1, 2, \ldots, mn\} \) is \( n \). Hence, the probability of failure if we do this procedure \( k \) times is:

\[
\frac{1}{m^k}
\]

so the probability of success is at least

\[
1 - \frac{1}{m^k}.
\]
Schwartz-Zippel Lemma

Claim. If $F \neq 0$ is a polynomial in $(x_1, x_2, \ldots, x_n)$ with $d_i$ the degree of $F(\cdot)$ in $x_i$ and $(I_1, I_2, \ldots, I_n)$ are finite subsets of elements in the domain of each variable, then the number of roots of $F$ in $I_1 \times I_2 \times \cdots \times I_n$ is at most:

$$\left( \sum_{i=1}^{n} \frac{d_i}{|I_i|} \right) \cdot \prod_{i=1}^{n} |I_i|$$
Claim. Schwartz-Zippel Lemma

• Proceed via induction. Base case is clear.
• Let $F'$ be the polynomial in $(x_2,\ldots,x_n)$ and suppose that $(y_2,\ldots,y_n)$ is not a zero of $F'$. Then $F(x_1,y_2,\ldots,y_n)$ has at most $d_1$ zeros.
• It follows that the number of roots cannot exceed

$$d_1 \cdot \prod_{i=2}^{n} |I_i| + \left( \sum_{i=2}^{n} \frac{d_i}{|I_i|} \right) \cdot \prod_{i=2}^{n} |I_i| \cdot |I_1|$$
Refresher...

• Define $B$ as follows:

$$b_{ij} = \begin{cases} 
  x_{ij} & \text{if } i \sim j, \ i \in U, \ j \in V \\
  0 & \text{otherwise}
\end{cases}$$

• **Claim**: $\det(B) \neq 0$ if and only if $G$ has a perfect matching.
Claim: $\det(B) \neq 0$ if and only if $G$ has a perfect matching.

- If the determinant is non-zero, there must be some $\prod_{i=1}^{n} b_{i\pi(i)}$ term that is non-zero.
- If there is a perfect matching, set those variables to 1 and the others to 0. Then, the determinant is not identically zero.
Now what?

• Computing the determinant is annoying. How would we combine terms, etc.?
• Instead, turn to randomized algorithm:
  - choose the $x_{ij}$ randomly from $\{1, \ldots, n^2 m\}$
  - compute $\det(B)$
  - if determinant is 0, continue this process until some confidence threshold is hit
How good is that?

- Via Schwartz-Zippel, there are at most

\[
\left( \sum_{i=1}^{n^2} \frac{1}{n^2 m} \right) \cdot \prod_{i=1}^{n^2} (n^2 m) = \frac{1}{m} \cdot \prod_{i=1}^{n^2} (n^2 m)
\]

roots of \( F(x_{11}, \ldots, x_{nn}) \) on \( \{1, \ldots, mn^2\}^{n^2} \). Hence, the probability of making an error is

\[
\frac{1}{m}.
\]
Probability of Success and Runtime

- It follows that with $k$ repetitions, we can say that we make an error in classification with probability at most

$$\frac{1}{m^k}$$

- The runtime of the determinant is $O(n^{2.376})$ and more practical methods could be worse.

- What about general graphs?
The Tutte Matrix

- Define $T$ as follows (why?):

$$
t_{ij} = \begin{cases} 
  x_{ij} & \text{if } i \sim j \text{ and } i > j \\
  -x_{ji} & \text{if } i \sim j \text{ and } i < j \\
  0 & \text{otherwise}
\end{cases}
$$

- **Claim**: $\det(T) \neq 0$ if and only if $G$ has a perfect matching.
Claim (i): If $G$ has a perfect matching, then $\det(T) \neq 0$

- Take a perfect matching $M$; for each pair $(i, j)$ we set $x_{ij} = 1$ if $(i, j) \in M$ and $i > j$ and otherwise set $x_{ij} = 0$.
- Take $\pi$ to be a permutation such that
  
  $i = \pi(j)$ and $j = \pi(i)$ for all $(i, j) \in M$

  which clearly exists (just swap pairs).

  
  $\Rightarrow \prod_{i=1}^{n} t_{i\pi(i)} \neq 0, \prod_{i=1}^{n} t_{i\pi'(i)} = 0 \Rightarrow \det(T) \neq 0$. 

Claim (ii): If \( \det(T) \neq 0 \) then \( G \) has a perfect matching.

- Suppose some \( \pi \) contains an odd cycle \( \sigma = (i_1, i_2, \ldots, i_r) \) with \( r \) odd. Then, consider some \( \pi' \) with \( \sigma' = (i_r, i_{r-1}, \ldots, i_1) \). It follows that

\[
\text{sgn}(\pi) \cdot \prod_{i=1}^{n} t_{i\pi(i)} = -\text{sgn}(\pi') \cdot \prod_{i=1}^{n} t_{i\pi'(i)}
\]

- Since the determinant is non-zero, there must be some permutation of only even cycles, so there must be a perfect matching.
Back to old tricks...

• Turn to randomized algorithm:
  - choose the $x_{ij}$ randomly from $\{1, \ldots, n^2 m\}$
  - compute $\det(T)$
  - if determinant is 0, continue this process until some confidence threshold is hit

• With $k$ repetitions, error w/ probability at most

$$\frac{1}{m^k}$$

• Runtime is again $O(n^{2.376})$
That’s Pretty Good

• This algorithm is easy to program (if we don’t want optimized runtime).

• For bipartite graphs, Hopcroft-Karp gives a runtime of \( O(E\sqrt{V}) = O(n^{2.5}) \).

• For general graphs, the Blossom method of Edmond’s was originally \( O(V^4) = O(n^4) \) but has been improved to \( O(E\sqrt{V}) = O(n^{2.5}) \).

• However, we haven’t found perfect matchings yet... 😞
Finding Perfect Matchings

• First approach: sequential algorithm.

- Pick random \((i, j) \in G\)
- Check if \(G \setminus \{i, j\}\) has a perfect matching
  - If so, put \((i, j) \in M\) and continue on \(G \setminus \{i, j\}\)
  - Else, continue on \(G - (i, j)\)
Finding Perfect Matchings

• Better approach: parallel algorithm. Won’t go into details (email me for them – it’s pretty cool).
  - Pick random weights for edges
  - Setting the x’s to $2^{w_i}$, the min weight matching will be the largest power of 2 dividing the determinant
  - Compute matchings by computing the determinant in parallel and for each edge, run through a process and output/don’t output the edge. Resulting output edges make that min weight maximum matching if one exists.
How Good Are Those?

• The first one is slow. (How slow?)
• The second one is good. It’s the same as finding if a perfect matching exists, assuming you can run in parallel.
• The second one succeeds with probability at least $\frac{1}{2}$, so you might need to run it a few (finite) times...
• Still, the second one with $O(n^{2.376})$ is a great bound
Final Remarks

• This method is easy to implement in code.
• It runs reasonably quickly, especially for finding if perfect matchings exist. Very quickly in the asymptotic/theoretical sense.
• We used randomized algorithms to get very good results.
• There isn’t much difference between bipartite and general graphs here.
END