6.890 Lecture 17 More on Matching
Scribe: Michael P. Kim **Date:** April 30, 2014
Edited by: Cynthia Day, Yinzhan Xu **Updated:** November 4, 2021

1 Introduction

In the last lecture, we discussed two algorithms for finding perfect matchings both of which leveraged a random substitution of the Tutte Matrix to find edges in a pefect matching. Recall the Tutte Matrix is defined as follows.

$$T(i,j) = \begin{cases} 0 & \text{if } (i,j) \notin E \\ x_{ij} & \text{if } i < j \\ -x_{ij} & \text{if } i > j \end{cases}$$

The naive algorithm uses the determinant of the Tutte Matrix as an oracle for whether a perfect matching exists, while the Rabin-Vazirani Algorithm finds edges to include in the matching by leveraging properties of the Tutte Matrix.

Recall the naive algorithm.

Algorithm 1: NaiveMatch(G)

for $e \in E$ **do**
  if $\det(T_{G\setminus\{e\}}) \neq 0$ **then**
    Remove e from G;

Because we can determine if the determinant of T is 0 with an $\tilde{O}(n^\omega)$ randomized algorithm, this gives us an $O(n^{\omega+2})$ benchmark. Now, recall the Rabin-Vazirani algorithm.

Algorithm 2: RV(G)

$T \leftarrow$ random substitution of Tutte Matrix mod some prime $p > n^3$;
$M \leftarrow \emptyset$;
while $|M| < \frac{n}{2}$ **do**
  $N \leftarrow T^{-1}$ `// run time bottleneck to invert T`
  Find j s.t. $N(1,j) \neq 0,\ T(1,j) \neq 0$;
  $M \leftarrow M \cup \{(1,j)\}$;
  $T \leftarrow T_{\{1,j\},\{1,j\}}$

Remember that the runtime of RV is $O(n^{\omega+1})$, primarily due to the bottleneck in the while loop to compute the inverse of the Tutte Matrix in each iteration. In this lecture, we will give two improvements. We will first improve Rabin-Vazirani to $O(n^3)$ by using a specialized algorithm to invert the Tutte Matrix in $O(n^2)$ time. Then, we give an $O(n^\omega)$ algorithm that mimics the naive algorithm, but selects edges in a smart order.

2 Improving Rabin-Vazirani to $O(n^3)$

Claim 1 (Mucha, Sankowski)**.** RV *can be implemented so that updating N takes $O(n^2)$ time.*

Corollary 2.1. *Perfect matchings can be found in $O(n^3)$ time.*

To see Claim 1, we will start by proving a lemma, which we will use to recompute the necessary parts of the Tutte Matrix in $O(n^2)$ time.

Lemma 2.1. *Let M be an $n \times n$ invertible matrix. Let $N = M^{-1}$. Let M and N be of the following form.*

$$M = \begin{array}{c|c|c} & k & n-k \\ \hline k & X & Z \\ \hline n-k & Y & W \\ \hline \end{array} \qquad N = \begin{array}{c|c|c} & k & n-k \\ \hline k & \widehat{X} & \widehat{Z} \\ \hline n-k & \widehat{Y} & \widehat{W} \\ \hline \end{array}$$

If $\widehat{X}$ is invertible, so is W, with $W^{-1} = \widehat{W} - \widehat{Y} \cdot \widehat{X}^{-1} \cdot \widehat{Z}$.

Proof. We know $M \cdot N = I$. This means that $Y\widehat{X} + W\widehat{Y} = 0$. By assumption, $\widehat{X}$ is invertible, so $Y = -W\widehat{Y}\widehat{X}^{-1}$. We also know that $Y\widehat{Z} + W\widehat{W} = I$. We can combine these facts to obtain the following result.

$$\begin{aligned} (-W\widehat{Y}\widehat{X}^{-1})\widehat{Z} + W\widehat{W} &= I \\ \implies W \cdot (\widehat{W} - \widehat{Y}\widehat{X}^{-1}\widehat{Z}) &= I \end{aligned}$$

Thus, W is invertible and its inverse is $\widehat{W} - \widehat{Y} \cdot \widehat{X}^{-1} \cdot \widehat{Z}$. □

Note that this lemma also holds for permutations of the columns/rows of M. In particular, this means we can apply the lemma to the Tutte matrix, where X is the 2×2 matrix formed by rows 1 and j and columns 1 and j and W is $T_{\{1,j\},\{1,j\}}$. WLOG, we'll assume $j = 2$. Then, if we let $\widehat{W} = N[3..n, 3..n] = N_{\{1,2\},\{1,2\}}$, $\widehat{Y} = N[3..n, \{1,2\}]$, $\widehat{X} = N[\{1,2\},\{1,2\}]$, $\widehat{Z} = N[\{1,2\}, 3..n]$, we will obtain

$$T^{-1}_{\{1,2\},\{1,2\}} = N_{\{1,2\},\{1,2\}} - N_{3..n,\{1,2\}} \cdot N[\{1,2\},\{1,2\}]^{-1} \cdot N[\{1,2\}, 3..n]$$

by Lemma 2.1. We know that $N[\{1,2\},\{1,2\}]^{-1}$ exists since $N[1,1] = N[2,2] = 0$ because N is skew-symmetric and $N[1,2] = -N[2,1] \neq 0$ because $j = 2$ was chosen to be such that $N[1,j] \neq 0$.

Additionally, we claim that we can compute this matrix inverse in $O(n^2)$ time. The inverse requires the subtraction of two $(n-2)\times(n-2)$ matrices which takes $O(n^2)$ time, and requires computing an $(n-2)\times(n-2)$ matrix by the multiplication of an $(n-2) \times 2$ matrix by a 2×2 matrix, and then a $(n-2) \times 2$ matrix by a $2 \times (n-2)$ matrix. These multiplications also take $O(n^2)$ time. Thus, with minor modifications to the Rabin-Vazirani Algorithm, we come up with an $O(n^3)$ time algorithm for finding a perfect matching. This algorithm can be further modified to compute perfect matchings in bipartite graphs in $O(n^\omega)$ time, but this result is not generalizable to other graphs.

3 Improving Naive Search to $O(n^\omega)$

Theorem 3.1 (Harvey)**.** *There is an $\tilde{O}(n^\omega)$ time algorithm for finding perfect matching in general graphs.*

The algorithm by Harvey reimplements the naive algorithm for perfect matching. The naive version uses the determinant of the Tutte Matrix as an oracle for whether a perfect matching exists, and incrementally removes unnecessary edges. The key idea to the improved version is to choose a way to access the edges such that checking if $G \setminus \{e\}$ has a perfect matching is cheap. Consider the following lemma.

Lemma 3.1. *Let M be an invertible $n \times n$ matrix and $S \subseteq [n]$ be a subset of the rows/columns. Let $\tilde{M}$ be an $n \times n$ matrix such that if $\tilde{M}(i,j) \neq M(i,j)$, then $i, j \in S$. Finally, let $\Gamma = I_{|S|} + (\tilde{M}[S,S] - M[S,S])M^{-1}[S,S]$. Then the following statements are true.*

1. *$\tilde{M}$ is invertible if and only if*

$$\det(\Gamma) \neq 0$$

2. *If $\tilde{M}$ is invertible, then its inverse is*

$$\tilde{M}^{-1} = M^{-1} - M^{-1}[:, S]\Gamma^{-1}(\tilde{M}[S, S] - M[S, S])M^{-1}[S, :]$$

An immediate corollary of *1* is that we can check if $\tilde{M}$ is invertible in $O(|S|^\omega)$ time. Also, if we're only interested in $\tilde{M}^{-1}[S, S]$, then we can compute this submatrix by using a $|S| \times |S|$ submatrix of M^{-1} in each submatrix in *2* of the lemma. In this case, the overall time to compute $\tilde{M}^{-1}[S, S]$ is $O(|S|^\omega)$ given M and M^{-1}.

Let $N = T^{-1}$. Let $\tilde{T}$ be the Tutte matrix corresponding to G with edge (i, j) removed. That is, $\tilde{T}$ is T with $\tilde{T}(i, j) = \tilde{T}(j, i) = 0$.

Claim 2. *Checking if* $\det(\tilde{T}) \neq 0$ *is in* $O(1)$ *time.*

To see this claim, note that we need to check if $\det(\Gamma) \neq 0$ where in this case Γ is as follows.

$$\begin{aligned}\Gamma &= \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} + \begin{bmatrix}0 & -T(i,j)\\ T(i,j) & 0\end{bmatrix} \cdot \begin{bmatrix}0 & N(i,j)\\ -N(i,j) & 0\end{bmatrix}\\ &= \begin{bmatrix}T(i,j)N(i,j)+1 & 0\\ 0 & T(i,j)N(i,j)+1\end{bmatrix}\end{aligned}$$

Thus, $G \setminus \{(i, j)\}$ has a perfect matching if and only if $T(i, j) \cdot N(i, j) \neq -1$.

Thus, we can detect the presence of a perfect matching quickly, and we have that $\tilde{T}$ is the substituted Tutte Matrix of $G \setminus \{(i, j)\}$. Now all that remains to be seen is how to recompute $\tilde{T}^{-1}$, the updated version of N. We could naively do this in $O(n^2)$ time, by the previous lemma but this would only give us an $O(n^4)$ algorithm. We need to do something more sophisticated to obtain the desired $O(n^\omega)$ run time.

Let $S_1, S_2 \subseteq V$ where $|S_1| = |S_2|$. Let's try removing edges from $S_1 \times S_2$ according to Algorithm 3.

Algorithm 3: DeleteCross(S_1, S_2)

```
if |S1| = |S2| = 1 then
    S1 = {s}, S2 = {r};
    if T(s,r) · N(s,r) ≠ −1 then
        T(s,r) = T(r,s) = 0    // remove (s,r)
        UpdateN;
        Return {(s,r)};
else
    S1 ← S11 ∪ S12,  S2 ← S21 ∪ S22    // partition S1 and S2 each into two equal subsets;
        R ← ∅
    for i, j ∈ {1,2} do
        U ← DeleteCross(S1i, S2j);
        UpdateN;
        R ← R ∪ U;
    Return R;
```

While this recursive procedure will work, the question that remains is how to update N efficiently. The solution will be to update only $N[S_1 \cup S_2, S_1 \cup S_2]$ after each DeleteCross call within DeleteCross(S_1, S_2).

In particular, we will maintain the invariant that within DeleteCross(S_1, S_2), $N[S_1 \cup S_2, S_1 \cup S_2]$ will be correct. Then at the base case when $|S_1| = |S_2| = 1$, we will have the correct values $T(s, r)$ and $N(s, r)$ and we can correctly figure out whether (r, s) can be deleted.

To update N after a call to DeleteCross(S_1, S_2), we have the old T before the call and the new $\tilde{T}$ of changes within the call which we can obtain by modifying T using the changes U that we got from the recursive call. All the changes will be in $(S_1 \cup S_2) \times (S_1 \cup S_2)$, so we can use Lemma 3.1 to update $N[S_1 \cup S_2, S_1 \cup S_2]$ in $O((|S_1| + |S_2|)^\omega)$ time. In order to do so, you have to maintain something that's not shown in the pseudocode, including the old value of $N[S_1 \cup S_2, S_1 \cup S_2]$ before calling DeleteCross(S_{1i}, S_{2j}), and all changes to T when calling DeleteCross(S_{1i}, S_{2j}), which are all easy to maintain.

Thus, the overall runtime of DeleteCross(S_1, S_2) is given by the following recurrence relation, where $n = |S_1| = |S_2|$.

$$\begin{aligned} & T(n) \leq 4T(\frac{n}{2}) + 4O(n^\omega) \\ \implies & T(n) = \tilde{O}(n^\omega) \end{aligned}$$

We also need to handle edges within a set S in order to appropriately partition it into S_1 and S_2. Consider the final algorithm.

(Below we assume that $|S|$ is a power of 2. If it is not, we can add a large enough matching Y of new nodes to the graph: the size of the graph at most doubles, and the new graph has a perfect matching if and only if the old one did.)

Algorithm 4: DeleteWithin(S)

if $|S| = 1$ **then**
  Return;
$S \leftarrow S_1 \cup S_2$ `// such that |S1| = |S2| = |S|/2`
$R \leftarrow \emptyset$;
$U \leftarrow$ DeleteWithin(S_1), UpdateN$(S, S), R \leftarrow R \cup U$;
$U \leftarrow$ DeleteWithin(S_2), UpdateN$(S, S), R \leftarrow R \cup U$;
$U \leftarrow$ DeleteCross(S_1, S_2), UpdateN$(S, S), R \leftarrow R \cup U$;
Return R;

The runtime of Algorithm 4 is given by the following recurrence relation.

$$\begin{aligned} T(n) &\leq 2T(\frac{n}{2}) + T(\text{DeleteCross}(\frac{n}{2})) \\ &= 2T(\frac{n}{2}) + T(\tilde{O}(n^\omega)) \\ \implies T(n) &= \tilde{O}(n^\omega) \end{aligned}$$

The final algorithm for finding a matching is

Algorithm 5: Matching(G)

$R \leftarrow$ DeleteWithin(V);
Return $E \setminus R$;

References

[1] Marcin Mucha and Piotr Sankowski, *Maximum Matchings via Gaussian Elimination*, FOCS, 248-255 (2004).

[2] Nicholas J.A. Harvey, *Algebraic Algorithms for Matching and Matroid Problems*, SIAM Journal on Computing, 39(2):679-702, (2009).