Matchings in Graphs

Lecturer: *Meena Mahajan*
Scribe: *Nitin Saurabh*

Meeting: 11
1st April 2010

So far, we have seen the following sequential algorithms for maximum matching.

1. Edmonds' Blossom Shrinking Algorithm, which runs in time $O(n^4)$.
2. Structural algorithm, which also runs in time $O(n^4)$.
3. Polyhedral algorithm, which runs in polynomial time but is much slower than the above two.

The fastest known deterministic sequential algorithms run in time $O(\sqrt{n}m)$, which are very old. There have been no improvement since then. So today we are going to look at randomized algorithms to construct a maximum matching in time $O(n^\omega)$, where ω is the exponent for the best known matrix multiplication algorithm.

$O(n^\omega)$ time bound, also, suffices for the following operations:

- Computing *Determinant* of an $n \times n$ matrix.
- Computing *Rank* of an $n \times n$ matrix.
- Computing *Inverse* of an $n \times n$ non-singular matrix.
- Computing *Maximum rank submatrix* of an $n \times n$ matrix

1 Randomized algorithm for perfect matching

Let $G = (V, E)$ be a graph, where $|V| = n$ and $|E| = m$. Let us define the *Tutte's Matrix* T of G as below:

$$T_{i,j} = \begin{cases} t_{i,j} \ , & \text{if } (i,j) \in E \text{ and } i < j \\ -t_{i,j} \ , & \text{if } (i,j) \in E \text{ and } i > j \\ 0 \ , & \text{otherwise} \end{cases}$$

T is a *skew-symmetric adjacency matrix.*

Theorem 1 (Tutte) *G has a perfect matching iff T is non-singular.*

Proof: See the proof of Claim 12 in Lecture 7. ■

Since $det(T)$ is a symbolic multivariate polynomial, it can have exponential length in n, and so computing it symbolically is inefficient. However, there are randomized identity tests for polynomials that just need to evaluate a polynomial at a random point.

Lemma 2 (Schwartz-Zippel) *If $p(x_1, x_2, \ldots, x_m)$ is a non-zero polynomial of degree d with coefficients in a field $\mathbb{F}$, then the probability that p evaluates to 0 on a random element $(f_1, f_2, \ldots, f_m) \in \mathbb{F}^m$ is at most $\frac{d}{|\mathbb{F}|}$.*

This yields the following monte carlo algorithm.

Algorithm 1 Self-reducibility algorithm

```
Choose t̃ ∈ F^m.
A = T(t̃) {substitute values for indeterminates}
if det(A) = 0 then
  reject
end if
for e ∈ E do
  if det(A|_{a_e=0}) ≠ 0 then
    Set a_e = 0
  end if
end for
Report E
```

The algorithm substitute random values for the indeterminates from the field $\mathbb{F}$. If A is not full rank then algorithm halts and reject. Otherwise, for each $e \in E$ it sets $a_e = 0$ and checks if $A|_{a_e=0}$ is full rank. If yes then permanently delete that edge, else restore the edge.

The Schwartz-Zippel lemma tells us that checking determinant equals zero fails with probability at most $\frac{n}{|\mathbb{F}|}$, where n is the degree of the polynomial $det(A)$. Using union bound,

$$Pr[Error] \leq (m+1) \cdot \frac{n}{|\mathbb{F}|}$$

This, also, points out that the size of the field $\mathbb{F}$ should be $O(n^3)$.

The time for each determinant calculation is $O(n^\omega)$, so the total time required by the algorithm is $n^\omega + (m \times n^\omega) \approx O(n^{\omega+2})$.

1.1 An Improvement

Fact 3 *Let M be a non-singular matrix and let $N = M^{-1}$. Let M' be a matrix which is identical to M except that $M'_{S,S} \neq M_{S,S}$, where $M_{S,S}$ is a submatrix containing rows S and columns S.*

1. *M' is non-singular iff $(I_{|S|} + (M'_{S,S} - M_{S,S}) \cdot N_{S,S})$ is non-singular.*
2. *If M' is non-singular, then*

$$M'^{-1} = N - N_{*,S} \cdot (I_{|S|} + (M'_{S,S} - M_{S,S}) \cdot N_{S,S})^{-1} \cdot (M'_{S,S} - M_{S,S}) \cdot N_{S,*}$$

Proof: Let $M_{S,S} = X$, $N_{S,S} = X'$ and $M'_{S,S} = Y$. Without loss of generality, we can consider

$$M = \begin{pmatrix} X & B \\ A & C \end{pmatrix} \qquad M^{-1} = N = \begin{pmatrix} X' & B' \\ A' & C' \end{pmatrix}$$
$$M' = \begin{pmatrix} Y & B \\ A & C \end{pmatrix}$$

Therefore,

$$MN = \begin{pmatrix} XX' + BA' & XB' + BC' \\ AX' + CA' & AB' + CC' \end{pmatrix} = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}$$

and

$$M'N = \begin{pmatrix} YX' + BA' & YB' + BC' \\ AX' + CA' & AB' + CC' \end{pmatrix}$$

But $AX' + CA' = 0$, and $AB' + CC' = I$. Therefore,

$$M'N = \begin{pmatrix} YX' + BA' & YB' + BC' \\ 0 & I \end{pmatrix}$$

We know that $det(N) \neq 0$. Hence,

$$\begin{aligned} det(M') \neq 0 &\Leftrightarrow det(M')det(N) \neq 0 \\ &\Leftrightarrow det(M'N) \neq 0 \\ &\Leftrightarrow det(YX' + BA')det(I) \neq 0 \text{ (Since, } det \begin{pmatrix} A & B \\ 0 & D \end{pmatrix} = det(A)det(D) \text{)} \\ &\Leftrightarrow det(YX' + BA') \neq 0 \\ &\Leftrightarrow det(YX' + I_{|X|} - XX') \neq 0 \text{ (Since, } XX' + BA' = I_{|X|}) \\ &\Leftrightarrow I_{|X|} + (Y - X)X' \text{ is non-singular} \end{aligned}$$

This proves Fact 3.1. To prove Fact 3.2, multiply M' on the left of the given expression for M'^{-1}. ■

Fact 4 *Let M be an $n \times n$ skew-symmetric matrix and is non-singular, then M^{-1} is also skew-symmetric.*

In Algorithm 1, we calculate determinant of the matrix, again from start, where essentially we have changed only two entries. Also, note that we only need to check whether determinant is zero or non-zero. These observations along with Fact 3 improves the run time as follows.

Let $N = A^{-1}$. Let $e = (i, j)$ be the edge to be deleted and $S = \{i, j\}$. By Fact 3, $A|_{a_e=0}$ is non-singular iff $(I_2 - A_{S,S} \cdot N_{S,S})$ is non-singular. This can be done in constant time. If e is not deleted then N does not change, else N must be updated. By Fact 3, N is updated to

$$N - N_{*,S} \cdot (I_2 - A_{S,S} \cdot N_{S,S})^{-1} \cdot (-A_{S,S}) \cdot N_{S,*}$$

This takes $O(n^2)$ time. Therefore, for each edge, algorithm decides if it can be deleted and, if so, then updates N. So the total time required is $2n^\omega + m(n^2) \approx O(n^4)$ and the probability of error is as before.

1.2 A Recursive algorithm

Now we will show how to use lazy updates to bring down the runtime to $O(n^\omega)$.

Algorithm 2 Recursive algorithm

Randomly instantiate *Tutte's Matrix* T to get A
if $det(A) = 0$ **then**
  reject
end if
$N = A^{-1}$
DeleteWithin(V)
Return E

Algorithm 3 DeleteWithin(S)

if $|S| = 1$ **then**
  return
end if
$S = S_1 \uplus S_2$ such that $|S_1| = |S_2|$ and $\uplus$ is a disjoint union
DeleteWithin(S_1)
Update $N(S, S)$
DeleteWithin(S_2)
Update $N(S, S)$
DeleteAcross (S_1, S_2)

Algorithm 4 DeleteAcross(R,S)

if $|R| = |S| = 1$ **then**
  $R = \{r\}$ and $S = \{s\}$
  if $A_{r,s} = 0$ **then**
    return
  else if $det(A|_{a_{r,s}=0}) \neq 0$ **then**
    $a_{r,s} = a_{s,r} = 0$
    Update $N[\{r, s\}, \{r, s\}]$
  end if
else
  $R = R_1 \uplus R_2$ and $S = S_1 \uplus S_2$ such that $|R_1| = |R_2| = |S_1| = |S_2|$
  for $i, j \in \{1, 2\}$ **do**
    DeleteAcross(R_i, S_j)
    Update $N[R \cup S, R \cup S]$
  end for
end if

Algorithm 2 does not update N all at once, instead, it only updates the parts of N that are needed in each recursive call. DeleteAcross(R, S) updates $N[R \cup S, R \cup S]$ after each recursive call to DeleteAcross(R_i, S_j). Hence $N[R \cup S, R \cup S] = A^{-1}[R \cup S, R \cup S]$ after DeleteAcross(R, S). Similarly, DeleteWithin(S) updates $N[S, S]$ after each recursive call to DeleteWithin(S_i). Therefore, whenever an edge, $e = (r, s)$, is deleted Algorithm 2 updates N to maintain $N_{r,s} = A^{-1}_{r,s}$ in each recursive call. Hence, correctness of the algorithm follows.

Let $A|_{a_e=0} = A'$. To Update$N[S, S]$ after call to DeleteWithin(S_1), $N(S, S)$ is set to (by Fact 3)

$$N_{S,S} - N_{S,S_1} \cdot (I + (A'_{S_1,S_1} - A_{S_1,S_1}) \cdot N_{S_1,S_1})^{-1} \cdot (A'_{S_1,S_1} - A_{S_1,S_1}) \cdot N_{S_1,S}$$

Similarly after recursively calling DeleteAcross(R_i, S_j), Update$N[R \cup S, R \cup S]$ is done as follows.

$$N_{R\cup S,R\cup S} = N_{R\cup S,R\cup S} - N_{R\cup S,R_i\cup S_j} \cdot (I + \triangle \cdot N_{R_i\cup S_j,R_i\cup S_j})^{-1} \cdot \triangle \cdot N_{R_i\cup S_j,R\cup S}$$

where $\triangle = A'_{R_i\cup S_j,R_i\cup S_j} - A_{R_i\cup S_j,R_i\cup S_j}$. Both these updates involve matrix multiplication and computing inverse of matrices of size at most $|S| \times |S|$. So, the time taken in each update is $O(|S|^\omega)$.

Runtime: Let $g(n)$ denote the time taken by DeleteAcross(R, S), where $|R| = |S| = n$. Then,

$$\begin{aligned} g(n) &= 4[g(n/2) + (2n)^\omega] \\ &= 4 \cdot g(n/2) + O(n^\omega) = O(n^\omega) \end{aligned}$$

Let $f(n)$ denote the time taken by DeleteWithin(S), where $|S| = n$. Then,

$$\begin{aligned} f(n) &= 2[f(n/2) + n^\omega] + g(n/2) \\ &= 2 \cdot f(n/2) + g(n/2) + O(n^\omega) = O(n^\omega) \end{aligned}$$

So, the total time required by Algorithm 2 is $2n^\omega + O(n^\omega) \approx O(n^\omega)$ and the probability of error is as before.

2 Randomized maximum matching Algorithm

If a graph does not have a perfect matching then Algorithm 2 rejects. But we can use Algorithm 2 to obtain a monte carlo algorithm for maximum matching as follows.

The correctness of Algorithm 5 follows, since a perfect matching in the reduced graph on full rank principal submatrix is a maximum matching in the original graph. The total time taken by Algorithm 5 is $2n^\omega + O(n^\omega) \approx O(n^\omega)$. The probability of error is as before.

Algorithm 5 Randomized maximum matching algorithm

Randomly instantiate *Tutte's Matrix* T to get A
Find the rank of A
Find the maximum rank principal submatrix of A
Use Algorithm 2 to find perfect matching, M, in the reduced graph on full rank principal submatrix
Output M.

3 Monte Carlo to Las Vegas

To make Algorithm 5 a Las Vegas algorithm we have to test if the output is indeed a maximum matching. One way is to compute Gallai-Edmonds decomposition, that is, $D(G)$, $A(G)$ and $C(G)$, where $A(G)$ is a witness set and then use the notion of witness set to show that the matching is indeed maximum.

- $D(G) = \{u | u \text{ is not critical } \}$
- $A(G) = \{u | u \notin D(G) \text{ but } u \text{ has a neighbour in } D(G)\}$
- $C(G) = V \setminus (A(G) \cup D(G))$

Consider graph G_u obtained from G by adding a new vertex u' adjacent only to u in G. Now, u is not critical iff (maximum matching in G_u = maximum matching in G + 1). So,

$$u \in D(G) \Leftrightarrow \text{ maximum matching in } G_u = \text{ maximum matching in } G + 1$$

For each $v \in V$, find whether v is critical. If not then put v in $D(G)$. At the end we have computed $D(G)$. To compute $A(G)$, the time required is $O(n + m)$. So, the time required to check whether the output of Algorithm 5 is indeed a maximum matching is $n \cdot n^{\omega} + O(n + m) \approx O(n^{\omega+1})$. With constant probability we succeed to compute Gallai-Edmonds decomposition. Hence the expected running time of the algorithm is $O(n^{\omega+1})$. This can be brought down to $O(n^{\omega})$ using algebraic techniques.