Advanced Graph Algorithms Jan-Apr 2014

Lecture 12 — January 29, 2014

Lecturer: Saket Saurabh *Scribe: Anantha Padmanabha MS*

1 Overview

In the last lecture, we proved that for an undirected graph $G(V, E)$ and its corresponding Tutte's matrix T and let $\hat{T}$ be the matrix obtained by substituting the variables in T by values for the variables from a large possible field S, then for any perfect-matching PM, some edge $(u, v) \in PM$ iff $\hat{T}^{-1}[u, v] \neq 0$.

In this class we shall design an algorithm of running time $O(n^{\omega})$ to find a perfect-matching for an undirected graph G.

2 The Bottleneck

Lets try to understand what is the bottleneck in finding such algorithm. Suppose inverse of a matrix can be obtained in $O(n^{\beta})$ time, then repeating this process for $O(n)$ times would yield us the perfect matching. The approach would be to select the first edge with corresponding non-zero entry in $\hat{T}^{-1}$ and removing the corresponding vertices from the graph and create new T by deleting the rows and coloumns of the selected vertices and adding this edge to the set perfect-matching edges. We can obtain the perfect matching by repeating this process for $\frac{n}{2}$ number of iterations. Thus the total time required will be $O(n^{\beta+1})$.

However, just to write the values of the entries of $\hat{T}^{-1}$ we need $O(n^2)$ time. Hence $\beta \geq 2$. Thus the total running time for finding the inverse would be $O(n^3)$. So in the new approach we must try to avoid computing the inverse of T as a whole at every step.

3 Useful tools

Theorem 1. *(Sherman-Morrison-Woodbury theorem) Let M be an $n \times n$ matrix, U be $n \times k$ matrix and V be $n \times k$ matrices also let $det(M) \neq 0$, then :*

- *$M + UV^t$ is non-singular iff $I + V^t M^{-1} U$ is non-singular.*
- *If $(M + UV^t)$ is non-sungular then :*

$$(M + UV^t)^{-1} = M^{-1} - M^{-1}U(I + V^t M^{-1} U)^{-1} V^t M^{-1}$$

Proof. (1) : Consider the following block multiplication of matrices:

$$\begin{bmatrix} I & V^t \\ -U & M \end{bmatrix} = \begin{bmatrix} I & V^tM^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} I + V^tM^{-1}U & 0 \\ -U & M \end{bmatrix}$$

$$Since, det(M) \neq 0$$

$$det(M + UV^t) = det(I + V^tM^{-1}U) \cdot det(M)$$

$$M + UV^t \text{ is non-singular iff } I + V^tM^{-1}U \text{is non-singular}$$

□

Computing $M^{-1} - M^{-1}U(I + V^tM^{-1}U)^{-1}V^tM^{-1}$ takes $O(n^2 f(k))$ time. Hence suppose we can express $\hat{T} = T + UV^t$ then we can compute its inverse in $O(n^2)$ time as $k = 2$ in this case. But however the total running time of the algorithm to find the perfect matching would be reduced to $o(n^3)$ which is not as good as $O(n^\omega)$

4 Driving idea for improving efficiency

Suppose M' is identical to M except that $M'(S, S) \neq M(S, S)$, where S is a set of size k Without loss of generality,

$M = \begin{bmatrix} M_{S,S} & M_{S,\bar{S}} \\ M_{\bar{S},S} & M_{\bar{S},\bar{S}} \end{bmatrix}$ and $M' = \begin{bmatrix} M'_{S,S} & M_{S,\bar{S}} \\ M_{\bar{S},S} & M_{\bar{S},\bar{S}} \end{bmatrix}$

Let, $\Delta_{S,S} = M'_{S,S} - M_{S,S}$

Also, $U = \begin{pmatrix} I \\ 0 \end{pmatrix}$ and V = $\begin{pmatrix} \Delta_{S,S} \\ 0 \end{pmatrix}$

Thus, $M' = M + UV^t$.

Lemma 2. *If $M' = M + UV^t$ and $M^{-1} = N$ then M' is non-singular iff $det(I + \Delta_{S,S} \cdot N_{S,S}) \neq 0$.*

Proof. By theorem 1, M' is non-singular iff $I + V^tM^{-1}U$ is non-singular.

$$I + V^tNU = I + \begin{pmatrix} \Delta_{S,S} & 0 \end{pmatrix} \begin{pmatrix} N_{S,S} & N_{S,\bar{S}} \\ N_{\bar{S},S} & N_{\bar{S},\bar{S}} \end{pmatrix} \begin{pmatrix} I \\ 0 \end{pmatrix}$$

$$I + V^tNU = I + \begin{pmatrix} \Delta_{S,S} \cdot N_{S,S} & \Delta_{S,S} \cdot N_{S,\bar{S}} \end{pmatrix} \begin{pmatrix} I \\ 0 \end{pmatrix}$$

$$I + V^tNU = I + \begin{pmatrix} \Delta_{S,S} \cdot N_{S,S} \end{pmatrix}$$

□

Note that checking whether $det(I + \Delta_{S,S} \cdot N_{S,S})$ is zero or not takes constant time because S is a set of fixed size.

Lemma 3. *If M' is non-singular then,*

$$M'^{-1} = N - N_{*,S}(I + \Delta \cdot N_{S,S})^{-1} \cdot N_{S,*}$$

Proof. bBy theorem 1, If $(M + UV^t)$ is non-sungular then :

$$(M + UV^t)^{-1} = M^{-1} - M^{-1}U(I + V^tM^{-1}U)^{-1}V^tM^{-1}$$

$$M'^{-1} = N - NU(I + \Delta_{S,S} \cdot N_{S,S})^{-1}V^tN$$

$$M'^{-1} = N - N \begin{pmatrix} I \\ 0 \end{pmatrix} (I + \Delta_{S,S} \cdot N_{S,S})^{-1} \begin{pmatrix} \Delta_{S,S} & 0 \end{pmatrix} N$$

$$M'^{-1} = N - N \begin{pmatrix} (I + \Delta_{S,S} \cdot N_{S,S})^{-1} \\ 0 \end{pmatrix} \begin{pmatrix} \Delta_{S,S} & 0 \end{pmatrix} N$$

$$M'^{-1} = N - N \begin{pmatrix} (I + \Delta_{S,S} \cdot N_{S,S})^{-1} & 0 \\ 0 & 0 \end{pmatrix} N$$

$$M'^{-1} = N - \begin{pmatrix} N_{*,S} \cdot (I + \Delta_{S,S} \cdot N_{S,S})^{-1} & 0 \end{pmatrix} N$$

$$M'^{-1} = N - N_{*,S}(I + \Delta \cdot N_{S,S})^{-1} \cdot N_{S,*}$$

□

Thus by using this approach, we can keep deleting the edges of the graph which are not part of any matching and update $\hat{T}$ accordingly. However still to know $N_{S,S}$, $\hat{T}^{-1}$ has to be computed which takes $O(n^2)$ time. So to increase the efficiency of the algorithm we need to avoid computation of $\hat{T}^{-1}$ at every step.

Consider a situation where we are sure that the change in the inverse will happen only to a portion of $\hat{T}^{-1}$. Then we can construct a pseudo-inverse which looks exactly like the inverse only at this portion. But we should make sure that whenever we access the next edge, it should be inside this portion only. Suppose we have a strategy to fix this portion at every step and we compute the pseudo-inverse of only that portion of the matrix, then we can avoid computation of the full inverse at every step and hence we can increase the time efficiency. Thus the strategy to do this would be to find inverse only of the part of the matrix as and when required. If we can somehow bound the number of computations of inverses and make sure that the query at every step accesses only the part of the matrix where the modified inverse is computed, then the running time will be reduced.

5 Algorithm :

This algorithm was proposed by Nicholas J. A. Harvey [1]

FindPerfectMatching(G)
Input: graph $G(V, E)$
Output: $E' \subseteq E$ **which forms perfect matching**
$T \leftarrow$ Tutte's matrix of G
.
$\hat{T} \leftarrow T$ replaced with variables from a large field S
$N \leftarrow T^{-1}$
DeleteEdgesWithin(V)
return remaining edges

DeleteEdgesWithin(S**)**
if $|S| = 1$ **then**
  return
else
  Partition $S = S_1 \dot{\cup} S_2$
  for *i = 1,2* **do**
    DeleteEdgesWithin(S_i)
  end
  DeleteEdgesCrossing(S_1, S_2)
end

DeleteEdgesCrossing(R, S**)**
if $|R| = 1$ **then**
  $r \in R, s \in S$
  if $\hat{T}(r, s) \neq 0$ *and* $N(r, s) \neq \frac{-1}{\hat{T}(r,s)}$ **then**
    set $\hat{T}(r, s) = \hat{T}(s, r) = 0$
    Update $N_{R \cup S, R \cup S}$
  end
else
  divide R and S into equal halfs : $R = R_1 \dot{\cup} R_2$ and $S = S_1 \dot{\cup} S_2$
  for $i = 1, 2$ **do**
    for $j = 1, 2$ **do**
      DeleteEdgesCrossing(R_i, S_j)
    end
  end
  Update $N_{R \cup S, R \cup S}$
end

Correctness of the algorithm : Let R and S are the partition of the vertex set $V(G)$ and let

$$E(R) = \{(u, v) \in E(G) | u, v \in R\}$$

$$E(S) = \{(u, v) \in E(G) | u, v \in S\}$$

$$E(R, S) = \{(u, v) \in E(G) | u \in R \text{ and } v \in S\}$$

Now, suppose $R = R_1 \dot{\cup} R_2$ and $S = S_1 \dot{\cup} S_2$ then,

$$E(S) = E(S_1) \dot{\cup} E(S_2) \dot{\cup} E(S_1, S_2)$$

$$E(R, S) = E(R_1, S_1) \dot{\cup} E(R_2, S_1) \dot{\cup} E(R_2, S_1) \dot{\cup} E(R_2, S_2)$$

The algorithm recursively partitions the vertex set in the above mentioned way and at the base case decides whether an edge can be deleted or not by checking the value of $N_{r,s}$. So if we prove that whenever the algorithm accesses $N_{r,s}$ if its value is exactly $\hat{T}^{-1}_{r,s}$ then the algorithm is correct.

The algorithm updates N whenever an edge is deleted, but however it does not update the whole of N at once. It updates N such that the following invariants hold :

1. $DeleteEdgesWithin(S)$ initially has $N_{S,S} = \hat{T}^{-1}_{S,S}$. It restores this call after each recursive call to $DeleteEdgesWithin(S_i)$ and after calling $DeleteEdgesCrossing(R, S)$
2. $DeleteEdgesCrossing(R, S)$ initially has $N_{R \cup S, R \cup S} = \hat{T}^{-1}_{R \cup S, R \cup S}$. it restores this property after deleting an edge and after each recursive call to $DeleteEdgesCrossing(R_i, S_j)$.

Suppose we prove that these invariants hold, then checking for the value of $N_{r,s}$ and deleting of the edge is done only at $DeleteEdgesCrossing(R, S)$. Since by invariant, $DeleteEdgesCrossing(R, S)$ initially has $N_{R \cup S, R \cup S} = \hat{T}^{-1}_{R \cup S, R \cup S}$ the algorithm checks for the actual value of $\hat{T}^{-1}_{r,s}$. So proving that the algorithm maintains the invariants suffices to prove that the algorithm is correct.

To prove that the invariant (1) holds, consider the execution of $DeleteEdgesWithin(S)$. Here, the Tutte's matrix changes whenever an edge is deleted. The edge can be deleted while recursively calling $DeleteEdgesWithin(S_i)$ or $DeleteEdgesCrossing(S_1, S_2)$.
Suppose the algorithm has just recursed on $DeleteEdgesWithin(S_1)$, let P be the Tutte's matrix before the call and Q be the Tutte's matrix after the recursion. Q might differ from P because the recursion might have deleted a few edges. However, P and Q differs only at $\Delta = Q_{S_1,S_1} - P_{S_1,S_1}$. During the deletion of edges the algorithm ensures that Q is always non-singular.

By lemma 3,

$$Q^{-1} = P^{-1} - P^{-1}_{*,S} \cdot (I + \Delta \cdot P^{-1}_{S1,S1})^{-1} \cdot \Delta \cdot Q^{-1}_{S1,*}$$

in particular for S,

$$Q^{-1}_{S,S} = P^{-1}_{S,S} - P^{-1}_{S,S} \cdot (I + \Delta \cdot P^{-1}_{S1,S1})^{-1} \cdot \Delta \cdot Q^{-1}_{S1,S}$$

If $N_{S,S}$ was the old value of this part before the call, to restore the new invariant new value for $N_{S,S}$ has to be computed. Thus the $Update$ function writes the new value for $N_{S,S}$ as :

$$N_{S,S} = N_{S,S} - N_{S,S1} \cdot (I + \Delta N_{S1,S1})^{-1} \cdot \Delta N_{S1,S}$$

Here the matrix multiplication takes $O(|S|^2)$ and computation of inverse takes $O(|S|^\omega)$ running time.
Similarly the recursive call and updation for $DeleteEdgesWithin(S_2)$ also takes place.

After calling $DeleteEdgesAccross(S_1, S_2)$, the invariant 2 ensures that

$$N_{S,S} = N_{S_1 \cup S_2, S_1 \cup S_2} = \hat{T}^{-1}_{S,S}$$

Thus both the invariants hold before and after the call of $DeleteEdgesWithin(S)$.

To prove that invariant (2) holds, consider the execution of $DeleteEdgesCrossing(S, R)$. The edges can be deleted while recursively calling $DeleteEdgesCrossing(R_i, S_j)$ or when $|R| = 1$.

Suppose $|R| = 1$, then either the edge is deleted or not deleted. If the edge is not deleted, then there is nothing to update. But if the edge is deleted, the corresponding $N_{r,s}$ has to be updated. Here, $\Delta = T_{r,s}$. So the update is done by:

$$N_{r,s} = N_{r,s} - N_{r,s} \cdot (\frac{1}{1 + P_{r,s} \cdot N_{r,s}}) \cdot T_{r,s} \cdot N_{r,s}$$

$$N_{s,r} = -N_{r,s}$$

After every the recurrance call of $DeleteEdgesCrossing(R_i, S_j)$ the following update is made :

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

where $\Delta = (Q - P)_{R_i \cup S_j, R_i \cup S_j}$
Thus the algorithm satisfies both the invariants.

Running time analysis : Let $f(n)$ and $g(n)$ respectively denote the running time of the functions DeleteEdgesWithin(S) and DeleteEdgesCrossing(R, S), where $n = |R| = |S|$. Updating N requires $O(|S|^\omega)$. Thus the recurrance is given by :

$$f(n) = 2f(\frac{n}{2}) + g(n) + O(n^\omega)$$

$$g(n) = 4g(\frac{n}{2}) + O(n^\omega)$$

By solving this recurrance we get $f(n) = O(n^\omega)$ and $g(n) = O(n^\omega)$

References

[1] Nicholas J. A. Harvey: Algebraic Algorithms for Matching and Matroid Problems. SIAM J. Comput. 39(2): 679-702 (2009)