Advanced Graph Algorithms Jan-Apr 2014

Lecture 22, 23 — April 05, 2014

Lecturer: Saket Saurabh *Scribe: Roohani Sharma*

1 Overview

In this lecture we study how to find the transitive closure of a directed graph in time $O(n^\omega)$. We also study a $O(n^\omega \log n)$ algorithm that computes all pair shortest distances. We then study an algorithm that addresses the problem of actually finding a shortest path between each pair of vertices.

2 Transitive Closure of a directed graph

We are given as input the adjacency matrix $A(D)$ of a directed graph $D(V, E)$. The aim is to find the transitive closure of D, i.e. the adjacency matrix $A(D^*)$ of a graph $D^*(V, E^*)$ such that for all $u, v \in V$, $(u, v) \in E^*$ if and only if there exist a path from u to v in D.

We use divide and conquer technique to achieve our goal. Let $V = V_1 \uplus V_2$ such that $|V_1| = |V_2| = \frac{n}{2}$.

$$\begin{array}{c} \\ V_1 \\ V_2 \end{array} \begin{array}{c} \begin{array}{cc} V_1 & V_2 \end{array} \\ \begin{pmatrix} & \\ & \end{pmatrix} \end{array}$$

Let $A(D^*) = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$ where E, F, G, H are submatrices defined as follows.

$E \in \{0,1\}^{V_1 \times V_1}$ and for all $i, j \in V_1$

$$E[i,j] = \begin{cases} 1 & \text{if there exists a path from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$$

$F \in \{0,1\}^{V_1 \times V_2}$ and for all $i \in V_1$ and $j \in V_2$

$$F[i,j] = \begin{cases} 1 & \text{if there exists a path from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$$

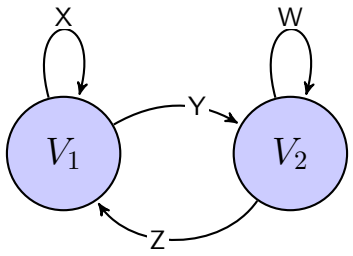
$G \in \{0,1\}^{V_2 \times V_1}$ and for all $i \in V_2$ and $j \in V_1$

$$G[i,j] = \begin{cases} 1 & \text{if there exists a path from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$$

$H \in \{0,1\}^{V_2 \times V_2}$ and for all $i, j \in V_2$

$$H[i,j] = \begin{cases} 1 & \text{if there exists a path from } i \text{ to } j \\ 0 & \text{otherwise} \end{cases}$$

Let us now model the paths between any two vertices in D as the following automata.



Here X, Y, W, Z are matrices as follows.

$X \in \{0,1\}^{V_1 \times V_1}$ and for all $i, j \in V_1$

$$E[i,j] = \begin{cases} 1 & \text{if there exists a path from } i \text{ to } j \text{ using vertices of } V_1 \text{ only} \\ 0 & \text{otherwise} \end{cases}$$

$F \in \{0,1\}^{V_1 \times V_2}$ and for all $i \in V_1$ and $j \in V_2$

$$F[i,j] = \begin{cases} 1 & \text{if } A(D)[i,j] = 1 \\ 0 & \text{otherwise} \end{cases}$$

$G \in \{0,1\}^{V_2 \times V_1}$ and for all $i \in V_2$ and $j \in V_1$

$$G[i,j] = \begin{cases} 1 & \text{if } A(D)[i,j] = 1 \\ 0 & \text{otherwise} \end{cases}$$

$H \in \{0,1\}^{V_2 \times V_2}$ and for all $i, j \in V_2$

$$H[i,j] = \begin{cases} 1 & \text{if there exists a path from } i \text{ to } j \text{ using vertices of } V_2 \text{ only} \\ 0 & \text{otherwise} \end{cases}$$

Suppose that recursively we have the matrices X and W (Y and Z can easily be constructed by looking up $A(D)$). Then E, F, G and H can be written in terms of X, Y, W, Z as follows.

($*$ denotes the operation of transitive closure on the matrix.)

$$E = (X + YW^*Z)^*$$

$$F = EYW^*$$

$$G = W^*ZE$$

$$H = W^* + GYW^*$$

Running time analysis

To get E, F, G and H let us see how many of each of the kinds of operations do we need.

1. # Transitive Closure of smaller $(\frac{n}{2} \times \frac{n}{2})$ matrices = 2

   This is because we need to calculate the transitive closure of W and then $(X + YW^*Z)$.

2. # Boolean Matrix Multiplications of $(\frac{n}{2} \times \frac{n}{2})$ matrices = 6

   For this first compute $P = (YW^*)$ and $Q = (W^*Z)$. Then $E = (X + PZ)^*, F = EP, G = QE, H = W^* + GP$.

In addtion to the above mentioned operations we also need to fill in entries for matrices Y and Z and perform boolean addition. Hence if $T(n)$ is the time taken to find transitive closure of a directed graph on n vertices and $BMM(n)$ is the time taken to multiply two $n \times n$ boolean matrices, then

$$T(n) \leq 2T(\frac{n}{2}) + 6BMM(\frac{n}{2}) + 2n^2$$

Since $BMM(n) = O(n^\omega) \leq cn^\omega$ for some constant c, we get

$$T(n) \leq 2T(\frac{n}{2}) + 6c(\frac{n}{2})^\omega + 2n^2$$

This recurrence solves to

$$T(n) = O(n^\omega)$$

.

3 All pair shortest distances

The all pair shortest distances problem is defined as follows.

We are given an undirected, unweighted, connected graph G. Let $V(G) = \{1, \ldots, n\}$ and let $A = (a_{ij})$ be the adjacency matrix of G i.e.,

$$a_{ij} = 1 \text{ if and only if } \{i, j\} \in E(G).$$

The aim is to compute the distance matrix $D = (d_{ij})$ of G such that

$$d_{ij} = \text{number of edges on a shortest path joining vertices } i \text{ and } j \text{ in } G.$$

3.1 The Strategy

Let us construct a graph $G^{'}$ (represented by its adjacency matrix B) from G as follows. The edge $\{i, j\} \in E(G^{'})$ if and only if either $\{i, j\} \in E(G)$ or there exists a two length path between i and j in G. Clearly if $G^{'}$ is a clique then G is a graph with diameter at most 2 and hence, $D = 2B - A$.

Otherwise $G^{'}$ has diameter at most $\frac{\delta}{2}$, where δ is the diameter of G. Then if $T = (t_{ij})$ is the all pair shortest distance matrix of $G^{'}$ then prove the following claim.

Claim 1. *For any pair i and j of vertices, if d_{ij} is even then $d_{ij} = 2t_{ij}$ and if d_{ij} is odd then $d_{ij} = 2t_{ij} - 1$.*

Proof. Let $d_{ij} = 2s$ for some s. Then if $i = i_0, i_1, \ldots, i_{2s-1}, i_{2s} = j$ is the shortest path in G then $i = i_0, i_2, \ldots, i_{2s-2}, i_{2s} = j$ is the shortest path between i and j in $G^{'}$ which has length s.

If $d_{ij} = 2s - 1$ and if $i = i_0, i_1, \ldots, i_{2s-2}, i_{2s-1} = j$ is the shortest path between i and j in G, then $i = i_), i_2, \ldots, i_{2s-4}, i_{2s-2}, i_{2s-1} = j$ is the shortest path between j and j in $G^{'}$ which has length s. □

Thus, if we can compute recursively the all pair shortest distance matrix, T of $G^{'}$ then by determining the parities of d_{ij} we can compute D - the all pair shortest distance matrix of G.

The following set of claims give us an efficient way to compute the parities of d_{ij}.

Claim 2. *Let i and j be a pair of distinct vertices in G. For any neighbour k of j in G we have $d_{ij} - 1 \leq d_{ik} \leq d_{ij} + 1$. Moreover there exists a neighbour k of j with $d_{ik} = d_{ij} - 1$.*

Proof. The proof follows trivially from the triangle inequality. □

Claim 3. *Let i and j be a pair of distinct verrices in G. If d_{ij} is even then $t_{ik} \geq t_{ij}$ for all neighbours k of j in G. If d_{ij} is odd then $t_{ik} \leq t_{ij}$ for all neighbours k of j in G, and $t_{ik} < t_{ij}$ for some neighbour k of j in G.*

Proof. Let d_{ij} be even, i.e. $d_{ij} = 2s$ for some s. Then $d_{ik} \geq 2s - 1$ for any neighbour k of j (from Claim 2). By Claim 1, $t_{ik} \geq s = t_{ij}$.

If d_{ij} is odd, i.e.$d_{ij} = 2s - 1$ for some s. Then $d_{ik} \leq 2s$ for any neighbour k of j (by Claim 1) and hence, $t_{ik} \leq s = t_{ij}$ (from Claim 2). Also, $d_{ik} = 2s - 2$ for some neighbour k of j (from Claim 2). For such a neighbour $t_{ik} = s - 1 < s = t_{ij}$. □

As a straightforward consequence of Claim 3 we have the following claim.

Claim 4. *d_{ij} is even if and only if $\sum_{k \in N_G(j)} t_{ik} \geq t_{ij}.degree_G(j)$ and d_{ij} is odd if and only if $\sum_{k \in N_G(j)} t_{ik} < t_{ij}.degree_G(j)$.*

Note that $\sum_{k \in N_G(j)} t_{ik} = \sum_{k \in [n]} t_{ik} a_{kj}$.

Hence, to efficiently compute the parities of d_{ij}'s , compute a matrix $X = TA$. Then

$$\text{if } x_{ij} \geq t_{ij}.degree(j) \text{ then } d_{ij} \text{ is even. / / if } x_{ij} < t_{ij}.degree(j) \text{ then } d_{ij} \text{ is odd.}$$

Hence, we have the algorithm **APD** described in the next subsection that takes as input the adjacency matrix A of G and returns the all pair shortest distance matrix D of G. The correctness of the algorithm **APD** follows from the above discussion.

Algorithm 1 APD(A)

Input: An undirected, unweighted, connected graph G with $V(G) = \{1, \ldots, n\}$ **Output:** The all pair shortest distance matrix $D = (d_{ij})$ of G such that d_{ij} = number of edges on the shortest path between i and j. **Description:**

$Z = A.A$
$B = (b_{ij})$ such that $b_{ij} = 1$ if $i \neq j$ and ($a_{ij} = 1$ or $z_{ij} > 0$) and $b_{ij} = 0$ otherwise.
if $b_{ij} = 1$ for all $i \neq j$ **then**
  return $D = 2B - A$;
end if
T=APD(B)
$X = T.A$
return $D = (d_{ij})$ where

$$d_{ij} = \begin{cases} 2t_{ij} & \text{if } x_{ij} \geq t_{ij}.degree(j) \\ 2t_{ij} - 1 & \text{if } x_{ij} < t_{ij}.degree(j) \end{cases}$$

3.2 Algorithm to compute all pair shortest distane matrix

Let us do the running time analysis of this algorithm. Let A be the adjacency matrix of the graph G on n vertices and diameter at most δ. Then,

$$T(n, \delta) = \begin{cases} cn^{\omega} + cn^2 & \text{if } \delta \leq 2 \\ 2cn^{\omega} + cn^2 + T(n, \frac{\delta}{2})t_{ij} & \text{if } \delta > 2 \end{cases}$$

Hence, running time of APD is $O(n^{\omega} \log n)$.

4 All pair shortest paths

Let us now address the problem of finding the shortest path for each pair of vertices in the grah. Note that we cannot compute all the shortest paths explicitly in $o(n^3)$ time as we have graphs with $\theta(n^2)$ pairs of vertices whose connecting paths have length $\theta(n)$. For example let G be the path graph on n vertices. For any i amongst the first $\frac{n}{4}$ vertices and any j amongst the last $\frac{n}{4}$ vertices, the smallest path connecting these two vertices has length atleast $\frac{n}{2}$. Hence, in G there are $\binom{\frac{n}{4}}{2} = \theta(n^2)$ pairs of vertices which have path lengths $\theta(n)$.

Hence in this section we do not find all shortest paths between any pair of vertices but compute a data structure that allows shortest paths to be reconstructed in time proportional to their lengths. We construct the data structure $S = (s_{ij})$ called the **shortest path successor matrix** where s_{ij} = a neighbour k of i that lies on a shortest path from i to j ($i \neq j$).

It is easy to see that once we have S we can reconstruct the shortest path from i to j as follows. Let $s_{ij} = k_1, s_{k_1,j} = k_2, s_{k_2,j} = k_3, \ldots, s_{k_p,j} = j$. Then the shortest path from i to j is $i, k_1, k_2, ldots, k_p, j$.

To compute S let us assume that we have an algorithm that takes two $n \times n$ boolean matrices

$A = (a_{ij})$ and $B = b_{ij}$, and outputs an integer matrix $W = (w_{ij})$ such that .

$$w_{ij} = \begin{cases} some \;\; k & \text{such that } a_{ik} = 1 \text{ and } b_{kj} = 1 \\ 0 & \text{if and only if no such } k \text{ exists} \end{cases}$$

This problem is called the boolean product witness matrix problem. Let $BPWM(A, B)$ return W.

We will now prove the following result.

Theorem 5. *For an unweighted, undirected, connected graph G, given its adjacency matrix A and its all pair shortest distance matrix D, we can compute the shortest path successor matrix S in time $3R(n) + O(n^2)$, where R(n) is the running time of BPWM.*

Proof. Let i and j be vertices with $d_{ij} = d > 0$. We want to compute s_{ij} = some k such that $(a_{ik} = 1)$ and $d_{kj} = d - 1$.

Let $B^{(d)} = ({b_{kj}}^{(d)})$ be a $n \times n$ boolean matrix such that $b_{kj} = 1$ if and only if $d_{ij} = d - 1$. Then let $W^{(d)} = BPWM(A, D^{(d)})$. Then for all i and j with $d_{ij} = d$,$s_{ij} = {w_{ij}}^{(d)}$.Hence, all the entries of S can be filled by computing $W^{(d)} = BPWM(A, B^{(d)})$ for all $1 \leq d \leq n - 1$.

We will now show that it sufices to call BPWM only thrice instead of $n - 1$. The idea behind reducing the number of function calls to BPMW is based on the following key observation.

Observation 6. *For any neighbour k of i, $d_{ij} - 1 \leq d_{kj} \leq d_{ij} + 1$. And if k lies on the shortest path between i and j then $d_{ik} = d_{ij} - 1$.*

Hence, to find s_{ij} it suffices to find a k such that $a_{ik} = 1$ and $d_{kj} \equiv_2 (d_{ij} - 1) mod 3$. Therefore, it is enough to call the BPWM function thrice as follows:

$$W^{(r)} = BPWM(A, C^{(r)}) \text{ for all } 0 \leq r \leq 2, \quad \text{where } C^{(r)} = ({c_{ij}}^{(r)}) \text{ is a } n \times n \text{ boolean matrix such that} \quad {c_{kj}}^{(r)} = 1 \text{ if and only if } (d_{kj} + 1) mod 3 = r.$$

Then $s_{ij} = {w_{ij}}^{(d_{ij} \;\; mod \;\; 3)}$. □

The procedure described in Theorem 5 is formalized in the next subsection.

4.1 Algorithm for computing the shortest successor matrix

The question of how BPWM finds the witness of two boolean matrices is discussed in the next lecture.

Algorithm 2 APSP(A,D)

Input: Adjacency matrix A of an undirected, unweighted, connected graph G with $V(G) = \{1, \ldots, n\}$ All pair shortest path matrix D of G **Output:** The shortest path successor matrix $S = (s_{ij})$ such that S_{ij} = some k such that k is a neighbour of i in the shortest path between i and j. **Description:**

for all $r = \{0, 1, 2\}$ **do**
  $D^{(r)} = ({d_{ij}}^{(r)})$ be the $n \times n$ boolean matrix with ${d_{ij}}^{(r)} = 1$ if and only if $(d_{ij} + 1) \mod 3 = r$
  $W^{(r)} = BPWM(A, D^{(r)})$
  return S such that $s_{ij} = {w_{ij}}^{(d_{ij} \mod 3)}$
end for