

11.1 Lecture Overview

- Seidel's algorithm
- Network Flow Problem

11.2 Seidel's algorithm

In this lecture, we consider the problem of finding shortest paths between all pairs of vertices in a undirected and unweighted graphs. Given a graph $G = (V, E)$, we shall compute the distance matrix (a matrix of shortest path lengths between any node pairs). Seidel's algorithm[S92] can solve the problem in $O(\mu(|V|) \log |V|)$ time, where $\mu(|V|)$ is time required to multiply two $n \times n$ matrices. When we use BFS, Breadth-first Search, the running time is $O(|V|^3)$.

11.2.1 Description of Seidel's algorithm

Given an undirected and unweighted graph G , defined over a set of vertices V and a set of edges E , G_2 is the undirected and unweighted graph which is constructed from G in the following manner: G is augmented by edges (i, j) if G has a path of length 2 from i to j . As shown in the figure 11.1, the solid edges are from G , the dotted edges are added in G_2 .

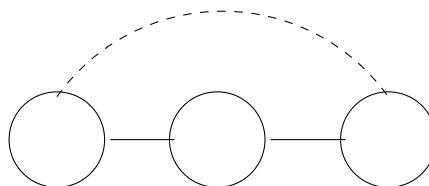


Figure 11.1: Augmenting graph G into G_2

We define

$M[G]$: the adjacency matrix of G ,

$D[G]$: the distance matrix for graph G .

$P[G]$: a binary matrix, where

$$P[G](i, j) = \begin{cases} 1 & \text{if shortest path from } i \text{ to } j \text{ has odd length} \\ 0 & \text{otherwise} \end{cases}$$

Seidel's algorithm is described as following:

Input: An undirected and unweighted graph $G = (V, E)$, where V is a set of vertices and E is a set of edges.

Output: $[D[G]]$ the distance matrix for graph G . of the shortest path from u to v .

The heart of Seidel's algorithms is the following procedure APSP, which, given the adjacency matrix of graph G , returns the distance matrix $D[G]$ for G . APSP computes $D[G]$ in time $O(\mu(|V|) \log |V|)$, if the time cost on recurrence is not included.

$D[G] = \text{APSP}(M[G])$

APSP($M[G]$): $D[G]$

1. **compute** $M[G_2]$
2. **if** $\forall i \neq j; M[G_2](i, j) = 1$
then
3. **return** $D[G](i, j) = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } M[G](i, j) = 1 \\ 2 & \text{otherwise} \end{cases}$
- else**
4. $D'[G] = \text{APSP}(M[G_2])$
5. **compute** $P[G]$
- endif**
6. **return** $D[G] = 2D'[G] - P[G]$

11.2.2 Proof of Correctness

Proof: Define the diameter of a graph G is the maximum shortest path from vertex i to vertex j ($\forall i \neq j$). The proof for the correctness of Seidel's algorithm can be shown using induction on diameter of G .

Base case: diameter ≤ 2 , thus $\forall i \neq j; M[G_2](i, j) = 1$, APSP stops at step 3.

Inductive Step: Assume the algorithm is correct when diameter is put to k , now consider a undirected graph G with diameter of $k + 1$. The expression in step 6 is true, as long as $\forall i, j; D[G](i, j) \leq k$. To any node pair (i, j) in G , we discuss $D[G](i, j)$ in two cases:

Case 1: if $P[G](i, j) = 0$, i.e. the path distance between i and j is even. This immediately implies, (and as is illustrated in the figure) that the path distance between i and j in G_2 is half of that in G , or

$$D[G_2]_{i,j} = \frac{1}{2} D[G]_{i,j}$$

Therefore $D[G]_{i,j} = 2D'[G] - P[G]$

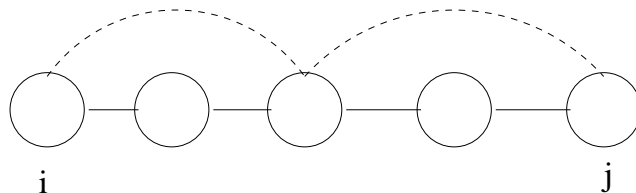


Figure 11.2: Path of even length

Case 2: if $P[G](i, j) = 1$, i.e. the path from (i, j) is of odd length. Let this be some length $2m + 1$ where m is a natural number. Then the path length to a node just before j would be $2m$, and can be actually reached in m steps using the edges in graph G_2 . Hence to reach j from this node, we need to take one extra step in G , giving us a final path length of $m + 1$. Thus, to generalize,

$$\begin{aligned}
 D[G_2](i, j) &= \frac{1}{2}[D[G](i, j) - 1] + 1 \\
 &= \frac{1}{2}D[G](i, j) + \frac{1}{2} \\
 &= \frac{1}{2}[D[G](i, j) + 1]
 \end{aligned}$$

and thus $D[G]_{i,j} = 2D'[G] - P[G]$

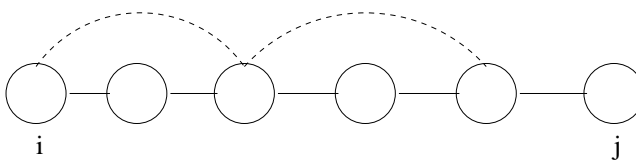


Figure 11.3: Path of odd length

Therefore, $D[G]_{i,j} = 2D'[G] - P[G]$, Step 6 and hence Seidel’s algorithm is true for graphs of all diameters, ■

11.2.3 Analysis of Seidel’s algorithm

In Step 1 and Step 5, matrix multiplication costs $O(\mu(|V|))$ time, we will discuss the implementation of Step 5 in the following section.

In Step 2,3 and Step 6, searching and updating the matrix take $O(|V|^2)$ time.

The length of diameter in G_2 is half of the length in original graph G . Since the diameter is at most $|V|$, the level of recurrence is $O(\log |V|)$. Clearly, When the recurrence stops, every connected component of G is a complete graph.

So that the total running time is $O(\mu(|V|) \log |V|)$.

11.2.4 Computing $P[G]$

Now we provide an efficient method to compute the parity matrix $P[G]$ in $O(\mu(|V|))$ time.

Claim: Define $X = D'[G] \cdot M[G]$, X is the product of $D'[G]$ and $M[G]$. $D'[G]$ is the distance matrix of G . We have the following claim:

$$P[G](i, j) = 0 \iff X(i, j) \geq D'[G](i, j) \cdot \text{degree}_{[G]}(j)$$

where, $\text{degree}_{[G]}(j)$ is the number of edges that are incident to node j in graph G .

Proof: In order to prove this claim, we need to show that both directions of the implication are true. Thus we can break down the proof in two parts:

- $P[G](i, j) = 0 \implies X(i, j) \geq D'[G](i, j) \cdot \text{degree}_{[G]}(j)$
- $P[G](i, j) = 1 \implies X(i, j) < D'[G](i, j) \cdot \text{degree}_{[G]}(j)$

The matrix X is computed in the following form:

$$\begin{aligned} X(i, j) &= \sum_{k=1}^{|V|} (D'[G](i, k) \cdot M[G](k, j)) \\ &= \sum_{k \in \text{Adj}_{[G]}(j)} (D'[G](i, k)) \end{aligned}$$

From this representation it is apparent that $X(i, j)$ is the sum of $\text{degree}_{[G]}(j)$ terms.

First, we prove that $P[G](i, j) = 0 \implies X(i, j) \geq D'[G](i, j) \cdot \text{degree}_{[G]}(j)$

Consider if each of these terms is in turn greater than $D'[G](i, j)$, then their sum would be greater than $D'[G](i, j) \cdot \text{degree}_{[G]}(j)$. Hence we only need to prove that if $P[G](i, j) = 0$, then

$$\forall k \in \text{Adj}_{[G]}(j), D'[G](i, k) \geq D'[G](i, j)$$

We will look at two cases:

Case 1: k is on the shortest path from i to j , which is shown in the following diagram as k_0 . It is apparent that the length of the path from i to j is equal to that of the path from i to k .

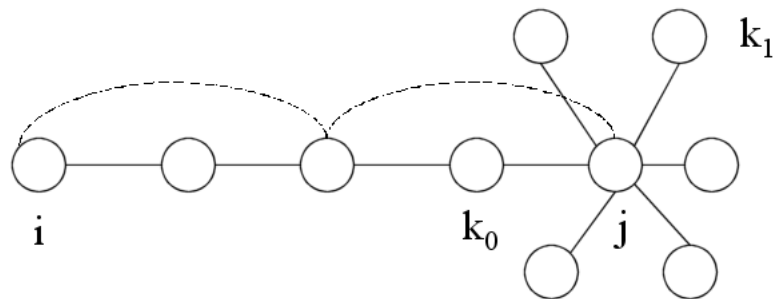
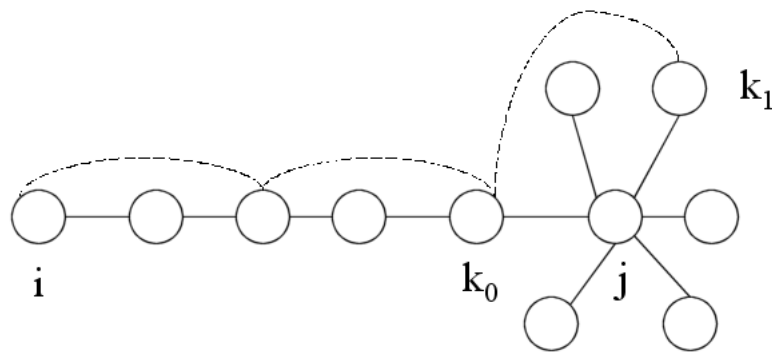
Case 2: k is not on any shortest path from i to j . such as k_1 in the following diagram, we can see that, since the number of edges between i and j is even, and k is adjacent to j , it would take at least one more step to reach k , hence the path from i to k is longer.

Thus, if $P[G](i, j) = 0$, then

$$X(i, j) = \sum_{k \in \text{Adj}_{[G]}(j)} (D'[G](i, k)) \geq D'[G](i, j) \cdot \text{degree}_{[G]}(j)$$

Therefore, $P[G](i, j) = 0 \implies X(i, j) \geq D'[G](i, j) \cdot \text{degree}_{[G]}(j)$

Second, we prove that: $P[G](i, j) = 1 \implies X(i, j) < D'[G](i, j) \cdot \text{degree}_{[G]}(j)$

Figure 11.1: An example for $P[G]_{(i,j)} = 0$ Figure 11.2: An example for $P[G]_{(i,j)} = 1$

We just need to prove that if $P[G]_{(i,j)} = 1$, then

$$\forall k \in \text{Adj}_{[G]}(j), D'[G]_{(i,k)} \leq D'[G]_{(i,j)}$$

and

$$\exists k \in \text{Adj}_{[G]}(j), D'[G]_{(i,k)} < D'[G]_{(i,j)}$$

We shall still look at two cases:

Case 1: k is on the shortest path (as k_0 in the above figure). We can easily see from the graph that $D'[G]_{(i,k)} < D'[G]_{(i,j)}$ since it can be reached directly using edges in G_2 , while we need to take an extra step to go to j , so that $D'[G]_{(i,k_0)} < D'[G]_{(i,j)}$.

Case 2: k is not on the shortest path (as k_1 in the above figure). Because the length between i and j is odd, in order to reach j we can use the edges from G_2 only till a node before j and then we must take an extra step on an edge in G . While, to reach k we can jump from this previous node to k using a G_2 edge directly. Hence the path lengths are equal.

Because there must exist a node $k_0 \in Adj_{[G]}(j)$ on the shortest path from i to j , so that when $P[G]_{(i,j)} = 1$, then

$$X_{(i,j)} = \sum_{k \in Adj_{[G]}(j)} (D'[G]_{(i,k)}) < D'[G]_{(i,j)} \cdot degree_{[G]}(j)$$

Then , $P[G]_{(i,j)} = 1 \implies X_{(i,j)} < D'[G]_{(i,j)} \cdot degree_{[G]}(j)$

■

11.3 Network Flow

We now look at a class of problems under the general heading of Network Flow. The following graph is an example of a flow network. It shows the capacity of each edge and the values (in parentheses) of a sample flow.

11.3.1 Description of Network Flow

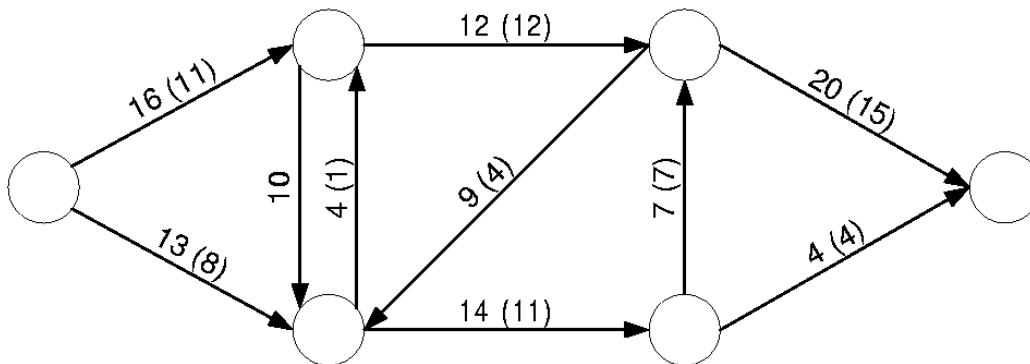


Figure 11.6: Example for a flow network

Input:

- Directed Graph $G, G = (V, E)$.
- Capacities: $C(u, v) > 0$ for $(u, v) \in E$
 $C(u, v) = 0$ for $(u, v) \notin E$
- A source node s , a sink node t

Output: The maximum flow from s to t with the flow function $f : V \times V \rightarrow \Re$ and the following properties:

1. Skew-symmetry: $f(u, v) = -f(v, u) \quad \forall u, v \in V$
2. Conservation of Flow: $\sum f(u, v) = 0 \quad \forall u, v \in (V - \{s, t\})$

3. Capacity Constraints: $f(u, v) \leq c(u, v) \quad \forall u, v \in V$
 $c(u, v) = 0$ if $(u, v) \notin E$

We define the size of a flow as the total flow coming out of the source:

$$\begin{aligned} |f| &= \sum_v f(s, v) \\ &= \sum_v f(v, t) \end{aligned}$$

11.3.2 The s - t cut

Define an s - t cut (abbreviated cut) of G is a partition of the vertices v into two sets A and B such that $s \in A$ and $t \in B$.

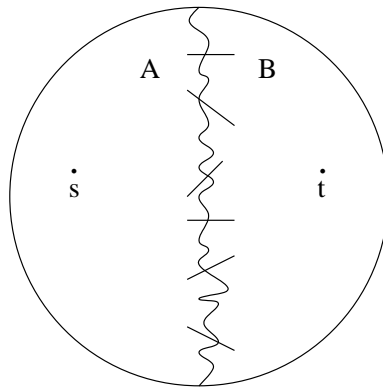


Figure 11.7: An example s - t cut

The capacity of cut (A, B) is defined as the sum of the capacities of all edges across the cut:

$$c(A, B) = \sum_{u \in A, v \in B} c(u, v)$$

The flow across a cut is defined similarly, as the flow across all edges across the cut:

$$f(A, B) = \sum_{u \in A, v \in B} f(u, v)$$

The capacity constraints still hold when talking about cuts, so we can say that $f(A, B) \leq c(A, B)$.

Claim 11.1 For any flow f and for all cuts (A, B) , $f(A, B) = |f|$

Proof: by induction on the size of A .

Base Case: $|A| = 1$ ($A = \{s\}$)

In this case, the claim follows from the definition of the size of a flow.

Inductive Step: We assume that $f(A, B) = |f| \quad \forall A$ such that $|A| \leq k$. In next lecture, we will show that it holds $\forall A$ such that $|A| = k + 1$. ■

References

- S92 R. SEIDEL, On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs, *Journal of Computer and System Sciences* **51**, 1995, pp. 400–403.
- CLR T. CORMEN and C. LEISERSON and R. RIVEST, Introduction to Algorithms, MIT Press, 1990
- SN10 HONGGANG ZHANG and S. JAISWAL Scribe notes for lecture 10 of Computer Science 611, University of Massachusetts at Amherst, 2000.
- SN11 M. BILLMERS and WEI LI Scribe notes for lecture 11 of Computer Science 611, University of Massachusetts at Amherst, 2000.