CMPSCI 611: Advanced Algorithms
Lecture 10: Seidel’s Algorithm

Andrew McGregor
Seidel’s Algorithm

Problem: For an undirected, unweighted graph $G$, compute all distances.

Seidel’s Algorithm is based on matrix multiplication and runs in time

$$O(\mu(n) \log n)$$

where $\mu(n)$ is the time to multiply two $n \times n$ matrices together. Recall

$$n^2 \leq \mu(n) \leq n^{2.3727}$$

Definition
Let $M_G$ be the adjacency matrix of $G = (V, E)$, i.e., an $n \times n$ binary matrix where

$$M_G(i, j) = 1 \text{ iff } (i, j) \in E$$
The $G_2$ graph

Definition
Given a undirected, unweighted graph $G = (V, E)$, define $G_2 = (V, E')$ where $(i,j) \in E'$ iff $\delta_G(i,j) \leq 2$.

Claim
If $M_{G_2}(i,j) = 1$ then $\delta_G(i,j) \leq 2$ and $\delta_G(i,j) = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } M_G(i,j) = 1 \\ 2 & \text{otherwise} \end{cases}$

Lemma

$$\delta_G(i,j) = 2\delta_{G_2}(i,j) - P_G(i,j)$$
where $P_G(i,j) = 1$ if $\delta_G(i,j)$ is odd and $P_G(i,j) = 0$ otherwise.

Proof.
Any path of length $2k$ in $G$ gives rise to a path of length $k$ in $G_2$. Any path of length in $2k + 1$ in $G$ gives rise to a path of length $k + 1$ in $G_2$. \qed
Seidel’s Algorithm

Algorithm (Seidel(\(M_G\)))

1. compute \(M_{G_2}\)
2. if \(\forall i \neq j : M_{G_2}(i, j) = 1\), return

\[
D_G[i,j] = \begin{cases} 
0 & \text{if } i = j \\
1 & \text{if } M_G(i, j) = 1 \\
2 & \text{otherwise}
\end{cases}
\]

3. else:
   3.1 compute \(D_{G_2} = \text{Seidel}(M_{G_2})\)
   3.2 compute \(P_G\)
   3.3 return \(D_G = 2D_{G_2} - P_G\)

Mystery Steps: How can we compute \(M_{G_2}\) and \(P_G\) quickly?
Depth of Recursion

- The diameter of a graph $G$ is the “longest shortest path”,
  \[
  \text{diam}(G) = \max \limits_{i,j} \delta_G(i,j)
  \]

- Note that if $\text{diam}(G) \geq 3$:
  \[
  \text{diam}(G_2) \leq \frac{\text{diam}(G)}{2} + \frac{1}{2} \leq \frac{2\text{diam}(G)}{3}
  \]

- After recursing $t$ steps, the diameter is at most
  \[
  (2/3)^t \text{diam}(G)
  \]
  and so after $\log(n/2)/\log(3/2)$ steps, the diameter is at most 2.
Computing $M_{G^2}$ via $M_G \times M_G$

Lemma

$$M_{G^2}(i,j) = \begin{cases} 
1 & \text{if } i \neq j \text{ and } (M_G(i,j) = 1 \text{ or } M^2_G(i,j) > 0) \\
0 & \text{otherwise}
\end{cases}$$

Proof.

$M^2_G(i,j) = \sum_k M_G(i,k)M_G(k,j) = \#$ of length 2 paths from $i$ to $j$. \qed

Can compute $M_{G^2}$ in $O(\mu(n))$ time.
Computing $P_G$ via $D_{G_2} \times M_G$

$P_G$ can be computed in $O(\mu(n))$ time...

**Lemma**

Let $X = D_{G_2} M_G$ where $D_{G_2}(i,j) = \delta_{G_2}(i,j)$. Then,

$$P_G(i,j) = 0 \iff \frac{X(i,j)}{\text{degree}_G(j)} \geq \delta_{G_2}(i,j)$$

where $\text{degree}_G(j)$ is the number of edges incident to node $j$ in graph $G$.

Note that,

$$\frac{X(i,j)}{\text{degree}_G(j)} = \sum_k \frac{\delta_{G_2}(i,k) M_G(k,j)}{\text{degree}_G(j)} = \sum_{k \in \text{Adj}_G(j)} \frac{\delta_{G_2}(i,k)}{\text{degree}_G(j)}$$

Fix $i$ and let $d_k = \delta_{G_2}(i,k)$, then we need to show:

$$P_G(i,j) = 0 \iff (\text{average of } d_k \text{ over neighbors } k \text{ of } j) \geq d_j$$
Proof of Lemma

If $P_G(i, j) = 0$, then $\delta_G(i, j) = 2d_j$

- For all neighbors $k$ note that $\delta_G(i, k)$ is either $2d_j - 1, 2d_j$, or $2d_j + 1$
- Hence, each $d_k$ is either $d_j$ or $d_j + 1$
- Therefore average $d_k$ values is at least $d_j$

If $P_G(i, j) = 1$, then $\delta_G(i, j) = 2d_j - 1$

- For all neighbors $k$ note that $\delta_G(i, k)$ is either $2d_j - 2, 2d_j - 1$, or $2d_j$
- Hence, each $d_k$ is either $d_j - 1$ or $d_j$
- At least one neighbor has $\delta_G(i, k) = 2d_j - 2$ and $d_k = d_j - 1$
- Therefore average $d_k$ values is strictly less than $d_j$
Total Running Time

Algorithm (Seidel($M_G$))

1. compute $M_{G_2}$
2. if $\forall i \neq j : M_{G_2}(i, j) = 1$, return

$$D_G(i, j) = \begin{cases} 
0 & \text{if } i \neq j \\
1 & \text{if } M_G(i, j) = 1 \\
2 & \text{if otherwise}
\end{cases}$$

3. else:
   3.1 compute $D_{G_2} = \text{Seidel}(M_{G_2})$
   3.2 compute $P_G$
   3.3 return $D_G = 2D_{G_2} - P_G$

Running Time: $O(\mu(n) \log n)$ since depth of recursion is $O(\log n)$ and each iteration takes $O(\mu(n))$ time.