

1 Introduction

In this lecture, we will talk about the problem of computing all-pairs shortest paths (APSP) using matrix multiplication. Formally, given a graph $G = (V, E)$, the goal is to compute the distance $d(u, v)$ for all pairs of nodes $u, v \in V$. Given that G has n nodes and m edges, it is easy to come up with algorithms that run in $\tilde{O}(mn)$ time¹ – for example, on graphs with nonnegative edge weights, we can run Dijkstra’s Algorithm from each node. When G is dense, this time is on the order of n^3 ; the natural question is can we do better?

For weighted graphs, the best known algorithm for APSP runs in $O\left(\frac{n^3}{2^{\Omega(\sqrt{\log n})}}\right)$ time, by Ryan Williams (2014). A major open problem is whether there exist “truly subcubic” algorithms for this version of APSP, namely, algorithms running in time $O(n^{3-\varepsilon})$ for some constant $\varepsilon > 0$.

For unweighted graphs, we know of algorithms that achieve this subcubic performance. In particular, for undirected graphs there is an algorithm running in $O(n^\omega \log n)$ time by Seidel (1992), and for directed graphs there is an algorithm running in $O(n^{2.575})$ time by Zwick (2002). This difference in runtime persists even with improvements in the matrix multiplication exponent, ω . For instance, if $\omega = 2$, then Seidel’s algorithm would run in $\tilde{O}(n^2)$ time, whereas Zwick’s algorithm would run in $\tilde{O}(n^{2.5})$ time. In this lecture, we will talk about the algorithms for unweighted graphs. In particular, we discuss a baseline algorithm using a hitting set which will work for directed or undirected graphs, and then describe Seidel’s algorithm for undirected graphs.

2 Hitting Set Algorithm

Given G , and particularly the adjacency matrix A which represents G , we would like to compute distances between all pairs of nodes. A natural first algorithm would be to compute the distances by successive boolean matrix multiplication of A with itself. The (i, j) th entry in A^k is 1 if and only if i has a path of length k to j . Thus, if the graph has finite diameter, then for all i, j , $d(i, j) = \min\{k \mid A^k[i, j] = 1\}$.

Fact 2.1. *If G has diameter D we can compute APSP in $O(Dn^\omega)$ time.*

If the diameter of G is small, then we have found a fast algorithm for computing APSP, but D can be $O(n)$ in which case we have no improvement from $O(n^3)$ run time. Nevertheless, we can compute all short distances less than some k in $O(kn^\omega)$ time, and then employ another technique to compute longer distances. The key idea is to use a “hitting set”.

Lemma 2.1. (*Hitting Set*) *Let S be a collection of n^2 sets of size $\geq k$ over $V = [n]$. With high probability, a random subset $T \subseteq V$ of size $O\left(\frac{n}{k} \log n\right)$ hits all the sets in S .*

With this hitting set lemma in mind, we can use Algorithm 1 to compute distances that are greater than or equal to k .

With high probability, this algorithm will compute distances $\geq k$ correctly (as these paths involve at least k nodes, so with high probability T hits the path). The algorithm requires running Dijkstra’s algorithm from $O\left(\frac{n}{k} \log n\right)$ nodes so takes $\tilde{O}\left(\frac{n}{k} n^2\right)$ time. If we use this algorithm to compute long distances and the iterative matrix multiplication to compute short distances, we have an algorithm for all-pairs shortest paths.

Theorem 2.1. *Let G be a directed or undirected graph on n nodes, with unit weights. APSP of G can be computed in $\tilde{O}(kn^\omega + \frac{n}{k} n^2)$ time.*

¹ $\tilde{O}(T(n)) = O(T(n) \cdot \text{polylog } n)$. In other words, the poly-logarithmic terms have been dropped.

Algorithm 1: LONGDIST(V, E)

Pick $T \subseteq V$ randomly s.t. $|T| = c \cdot \frac{n}{k} \log n$ for large enough constant c
foreach $t \in T$ **do**
 | Compute DIJKSTRA(t)
foreach $u, v \in V$ **do**
 | Compute $d(u, v) = \min_{t \in T} d(u, t) + d(t, v)$

When we optimize for a choice of k and set it to $n^{(3-\omega)/2}$, the runtime comes out to be $\tilde{O}\left(n^{\frac{3+\omega}{2}}\right)$ which is roughly $\tilde{O}(n^{2.69})$.

3 Seidel's Algorithm

While this first algorithm gives us a fast algorithm for computing APSP, the question remains, can we do better? In particular, can we avoid computing short and long distances separately? Can we leverage matrix multiplication to compute all the shortest paths?

In fact, we can improve on the hitting set algorithm for undirected graphs as follows. Given a graph G with adjacency matrix A , consider its boolean square $A^2 = A \cdot A$, where \cdot represents boolean matrix multiplication. Consider a graph G' with adjacency matrix $A' = A^2 \vee A$.

Fact 3.1. $d_{G'}(s, t) = \left\lceil \frac{d(s, t)}{2} \right\rceil$

To see this fact, note that edges in A^2 represent paths of length 2 in the original graph G , and G' also contains the edges of G . Thus, any path of length $2k$ in G induces a path of length k in G' using only edges of A^2 , and also any path of length $2k + 1$ induces a path of length k (from A^2) followed by a single original edge, thus forming a path of length $k + 1$.

Now suppose that we have a way of determining the parity of the distance between all pairs of nodes. Then we can use the following recursive strategy to compute APSP.

Algorithm 2: APSP Idea

Given an adjacency matrix A
Compute $A^2 \vee A$
Recursively compute $d' \leftarrow \text{APSP}(A^2 \vee A)$
foreach $u, v \in V$ **do**
 | **if** $d'(u, v)$ *is even* **then**
 | $d(u, v) = 2d'(u, v)$
 | **else**
 | $d(u, v) = 2d'(u, v) - 1$

Note that in each recursive call, the diameter of the graph decreases by 2, and that after $\log n$ iterations, A will be the all 1s matrix with 0s along the diagonal, which we can detect. Thus, if we can find a way to determine the parity of a u, v -path efficiently, we should obtain an efficient recursive algorithm for APSP.

Consider any pair of nodes $i, j \in V$ and another node which is a neighbor of j , $k \in N(j)$. By the triangle inequality (which holds in unweighted, undirected graphs), we know $d(i, j) - 1 \leq d(i, k) \leq d(i, j) + 1$.

Claim 1. *If $d(i, j) \equiv d(i, k) \pmod{2}$, then $d(i, j) = d(i, k)$.*

Proof. By the triangle inequality, $d(i, j)$ and $d(i, k)$ differ by at most 1. Thus, if their parity is the same, they must also be equal. \square

Claim 2. Let $d_{G^2}(i, j)$ be the distance between i and j in G^2 defined by $A^2 \vee A$. Then, (a) if $d(i, j)$ is even and $d(i, k)$ is odd then $d_{G^2}(i, k) \geq d_{G^2}(i, j)$. (b) If $d(i, j)$ is odd and $d(i, k)$ is even, $d_{G^2}(i, k) \leq d_{G^2}(i, j)$ and there exists a $k' \in N(j)$ such that $d_{G^2}(i, k') < d_{G^2}(i, j)$.

Proof of (a).

$$d_{G^2}(i, j) = \frac{d(i, j)}{2}$$

$$d_{G^2}(i, k) = \frac{d(i, k) + 1}{2} \geq \frac{d(i, j)}{2}$$

so $d_{G^2}(i, k) \geq d_{G^2}(i, j)$. □

Proof of (b).

$$d_{G^2}(i, j) = \frac{d(i, j) + 1}{2} \geq \frac{d(i, k)}{2} = d_{G^2}(i, k)$$

so in general, $d_{G^2}(i, k) \leq d_{G^2}(i, j)$, and for the neighbor of j along the shortest path from i to j , which we call k' , we know $d(i, k') < d(i, j)$. □

Claim 3. If $d(i, j)$ is even, then

$$\sum_{k \in N(j)} d_{G^2}(i, k) \geq \deg(j) d_{G^2}(i, j)$$

and if $d(i, j)$ is odd, then

$$\sum_{k \in N(j)} d_{G^2}(i, k) < \deg(j) d_{G^2}(i, j)$$

This third claim follows directly from the first two. Additionally, if we can compute the sums here in $O(n^\omega)$ time, then the overall runtime will be $O(n^\omega \log n)$ as desired. The right expression can be computed in $O(n^2)$ time, which will be subsumed by the $O(n^\omega)$ term.

Consider D , an $n \times n$ matrix where $D(i, j) = d_{G^2}(i, j)$. We want to decide for each i, j pair whether $\sum_{k \in N(j)} d_{G^2}(i, k) < \deg(j) d_{G^2}(i, j)$. Consider the integer matrix product DA . Note that

$$(D \cdot A)[i, j] = \sum_{k \in N(j)} d_{G^2}(i, k)$$

so this matrix product allows us to compute the left expression.

Now we are ready to state Seidel's Algorithm in full.

Claim 4. *Seidel's Algorithm runs in $O(n^\omega \log d)$ time where d refers to the diameter of the graph.*

Proof. The run time can be expressed as the following recurrence relation.

$$T(n, d) \leq T(n, \frac{d}{2}) + O(n^\omega)$$

$$\implies T(n, d) \leq O(n^\omega \log d)$$

Because $d \leq n$, this run time is upper bounded by $O(n^\omega \log n)$. □

Note that Seidel's Algorithm relies on fast integer matrix multiplication, which runs in $O(n^\omega)$, but for which no known fast combinatorial algorithms exist. Some questions remain open whose answers could speed up the computation of APSP in theory and in practice: Is the integer matrix multiplication step avoidable? Are there fast combinatorial matrix multiplication algorithms over the integers?

Algorithm 3: SEIDEL(A)

```
if  $A$  is all 1s except the diagonal then
| return  $A$ 
else
| Compute boolean product  $A^2$ 
|  $D \leftarrow \text{SEIDEL}(A^2 \vee A)$ 
| Compute integer product  $D \cdot A$ 
|  $R \leftarrow 0^{n \times n}$ 
| foreach  $i, j \in V$  do
|   | if  $DA(i, j) < \text{deg}(j)D(i, j)$  then
|     |  $R(i, j) \leftarrow 2D(i, j) - 1$ 
|     else
|       |  $R(i, j) \leftarrow 2D(i, j)$ 
| return  $R$ 
```

References

- [1] Raimund Seidel, *On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs*, Journal of Computer and System Sciences 51, pp. 400-403 (1995).