CS 388R: Randomized Algorithms Fall 2017

Lecture 11 — Oct. 5, 2017

Prof. Eric Price *Scribe: Prateek Kolhar, Matt Jordan*

NOTE: THESE NOTES HAVE NOT BEEN EDITED OR CHECKED FOR CORRECTNESS

1 Overview

In this lecture, we look at the problem of finding the shortest paths between all nodes in a graph. We will first briefly look at some deterministic algorithms to achieve this and then look at certain randomized strategies.

Some standard deterministic algorithms:

Algorithm	Sources	Negative Weight	Time
Dijkstra	Single	No	$O(m + n \log n)$
Bellman-Ford	Single	Yes	$O(mn)$
Floyd-Warshall	All Pairs	Yes	$O(n^3)$

Floyd-Warshall Algorithm is the simplest to implement with the following pseudo code:

Data: Distance matrix D
Result: Shortest path matrix D
for $k \in [n]$ **do**
  for $i \in [n]$ **do**
    for $j \in [n]$ **do**
      $D_{ij} = \min(D_{ij}, D_{ik} + D_{kj})$
    end
  end
end

2 Faster Algorithm using Matrix Multiplication

The elements of the output matrix in matrix multiplication operation can be written down this way:

$$(AB)_{ij} = \sum_k A_{ik} \times B_{kj}$$

In this operation if we replace $(\sum, \times)$ with $(min, +)$ we essentially get the Floyd-Warshall Algorithm. And by doing so, all shortest paths can be computed in the same time complexity as matrix multiplication.
Matrix multiplication algorithms proposed in the past:

- Naive: $O(n^3)$
- Strassen '69: $O(n^{2.8074})$
- Coppersmith & Winograd '89: $O(n^{.375477})$
- Strothers '10: $O(n^{2.374})$
- Vassilevska-Williams '11: $O(n^{2.372873})$
- A better lower bound for matrix multiplication is still an open problem

In general, the time complexity of matrix multiplication is represented as $O(n^\omega)$. Our goal is to leverage some of these faster matrix multiplication techniques in finding shortest paths.

2.1 Naive Method

Consider A, the adjacency matrix, then A^2_{ij} is the number of paths from i to j of length 2. And, A^l_{ij} is the number of length l paths from i to j. Adding the identity matrix to A acts as if we added self loops to the graph, so then A^l_{ij} gives the paths for length $\leq l$.

To get the lengths of all pair shortest paths we just compute:

$$A^1, A^2, A^3, ..., A^n$$

and set the path length:

$$D_{ij} = \operatorname*{argmin}_k I[(A^k)_{ij} = 1]$$

$$I[*] \rightarrow \text{indicator function}$$

The time complexity is $O(n \cdot n^\omega) \simeq O(n^{3.373})$. This is worse than Floyd-Warshall algorithm.

2.2 Approximation

Suppose we want a 2-approximation of D_{ij}, which is X_{ij} such that $D_{ij} \in [\frac{X_{ij}}{2}, X_{ij}]$. We can compute:

$$A^1, A^2, A^4, A^8, ..., A^n$$

in $O(n^\omega) \log(n))$ time by repeatedly squaring.

Now consider the graph formed by using A^2 adjacency matrix. This is the graph with all length 2 paths as new edges. Let $D^{'}$ be the distance between all pairs in this graph. Our goal is to find D from $D^{'}$ and A in $O(n^\omega)$ time. When you compare the A graph with the A^2 graph there are 2 cases possible:

- if D_{ij} is even then $D^{'}_{ij} = \frac{D_{ij}}{2}$
- if D_{ij} is odd then $D^{'}_{ij} = \frac{D_{ij}+1}{2}$

So, we need to calculate $D_{ij} \mod (2)\ \forall i, j$ from $D^{'}$ and A. Lets look at the following 2 cases for nodes around the neighborhood, $N(i)$, of node-i:

- If D_{ij} is even then $\forall u \in N(i), D^{'}_{uj} \in \{D^{'}_{ij}, D^{'}_{ij}+1\}$ and for at least 1 $u \in N(i)$, we have $D^{'}_{uj} = D^{'}_{ij}$
- if D_{ij} is odd then $\forall u \in N(i), D^{'}_{uj} \in \{D^{'}_{ij}, D^{'}_{ij}-1\}$ and for at least 1 $u \in N(i)$, we have $D^{'}_{uj} = D^{'}_{ij} - 1$

This can be clearly seen from the fact that if the distance from i to j in A is even $(2l)$ then the neighbor u of i is at a distance of only $2l-1$, $2l$ or $2l+1$. In A^2, the distance from i to j is l and from u to j is l or $l+1$. If $D_{uj} = 2l-1$ in A, then it still takes l steps from u to j in A^2, A similar argument can be made for the case when D_{ij} is odd. Coming back to the original problem of reconstructing D_{ij}, we sum up the distances over the neighborhood of i:

- if D_{ij} is even then $\sum_{u \in N(i)} D^{'}_{uj} > D^{'}_{ij} \cdot |N(i)|$
- if D_{ij} is odd then $\sum_{u \in N(i)} D^{'}_{uj} < D^{'}_{ij} \cdot |N(i)|$

We can express these sums in matrix multiplication form as:

$$\sum_{u \in N(i)} D^{'}_{uj} = \sum_{u \in [n]} A_{iu} D^{'}_{uj} = (AD^{'})_{ij}$$

We compare $AD^{'}$ to $D^{'}|N(i)|$ to get $D_{ij} \mod (2)$ and set

$$D = 2D^{'} - (D \quad mod \quad 2)$$

This takes n^{ω} time for each step and a total time of $O(n^{\omega} \log(n))$

3 Determining shortest paths

In the last section we discussed how to compute the lengths of all pairs shortest paths, which we summarized in the matrix D. Note that D says nothing about what the paths are. Suppose we're given D and A; we want an efficient algorithm for finding the successor matrix S such that S_{ij} is k when the shortest path from node i to node j looks like $i \to k \to ... \to j$. This will allow us to determine shortest paths in time proportional to path length.

3.1 Easy case

Let's start with an easy case: let G be tripartite composed of a left, middle and right set. Let A refer to the adjacency matrix between the left and middle sets, and B refer to the adjacency matrix between the middle and right sets. Observe that the number of middle nodes k such that $i \to k \to j$ is a path is equivalent to the (i, j)'th entry of AB:

$$(AB)_{ij} = \sum_k A_{ik} B_{kj} = \#\text{ of middle nodes}$$

To make things easy, suppose only one such middle node, k^* exists, and our goal is to identify *which* node it is. Define A' such that $A'_{ik} = k \cdot A_{ik}$, so $(A'B)_{ij} = k^*$. Thus we can identify the intermediate node for a path in $O(n^\omega)$ time. We say that this intermediate node, k is a witness for the product of AB.

3.2 Easy-ish case

Suppose now that there are exactly r witnesses $k_1, k_2, ..., k_r$ such that $i \to k_d \to j$ is a path for all $d \in [r]$. Our technique from the easy case will no longer work, because $(A'B)_{ij}$ as defined above wouldn't allow us to determine a particular k_d. The idea here is to delete **all but one** of these $i \to k_d$ edges so we only end up with one witness, or rather, delete each edge **independently** with probability $1 - \frac{1}{r}$.

Define $A'_{ik} = A_{ik} \cdot k \cdot Z_k$ where Z_k is a Bernoulli random variable with probability $\frac{1}{r}$. Then the probability that exactly one witness k^* remains is

$$r \cdot (1 - \frac{1}{r})^{r-1} \cdot \frac{1}{r} = (1 - \frac{1}{r})^{r-1} > \frac{1}{e}$$

So now we just need to repeat this procedure $O(\log n)$ and then we have exactly one witness at least once with high probability. This approach has runtime $O(n^\omega \log n)$.

3.3 Medium case

What happens if now there are many different r's? That is, the number of intermediate nodes is not constant across our choice of source node i? We don't want to try the approach used in the 'easy-ish case' with all possible r, but instead we can try r to be powers of 2: $r = 1, 2, 4..., n$.

Suppose for a given i, the true number of intermediate nodes is r^*. Then when we let r be such that $r^* \leq r \leq 2r^*$, meaning we delete edges in A with probability $\frac{1}{r}$. Then the probability that exactly one witness remains is

$$\mathbb{P}\left[1 \text{ witness remains}\right] = r^* \left(1 - \frac{1}{r}\right)^{(r^*-1)} \frac{1}{r} \geq r^* \frac{1}{2r^*} \frac{1}{e} \geq \frac{1}{2e}$$

So if we run each choice of r $O(\log n)$ times then with high probability we find a witness for all i, j. Since there are $O(\log n)$ choices of r, and each step requires $O(n^\omega)$ time, then the total runtime is $O(n^\omega \log^2 n)$.

3.4 Hard Case

Now we're ready to extend the techniques we used in tripartite graphs to general (non-tripartite) graphs. Recall our goal: for all i, j we want to find a k such that $A_{ik} = 1$ and $D_{kj} = D_{ij} - 1$.

The idea here is to find the successor matrix for all paths of length l, $l - 1$, ... 1. We can do this by defining a matrix $R^{(l)}$ to be an $n \times n$ $0 - 1$ matrix:

$$R^{(l)}_{ij} = \begin{cases} 1 & \text{if } D_{ij} = l - 1 \\ 0 & \text{otherwise} \end{cases}$$

Suppose that the shortest path from i to j is of length l. Then k is a witness for this path if and only if it is one of the witnesses for $AR^{(l)}$. This follows because if k is a witness for the $i \to j$ path, then $D_{kj} = l - 1$ so both $A_{ik} = 1$ and $D_{kj} = 1$, which is to say that $(AR^{(l)})_{ij} = 1$ which is the same as k being a witness for $AR^{(l)}$. We can find witnesses for $AR^{(l)}$ with high probability using the technique described in the 'medium case' above in $O(n^\omega \log^2 n)$ time.

However the length of the shortest path between any two nodes can adopt n different values, so if we were to use the above strategy, we'd have to define n different $R^{(l)}$ matrices. Recall from our deterministic technique to find D that for any neighbor k of node i, $D_{ij} - 1 \leq D_{kj} \leq D_{ij} + 1$. And note that any k such that $D_{kj} = D_{ij} - 1$ is a successor for $i \to j$. So as long as $D_{kj} \equiv D_{ij} - 1 \mod 3$, k is a successor.

Instead of having to compute $R^{(l)}$ for each $l \in [n]$, we only need to compute three $R^{(0)}$:

$$R^{(0)}_{ij} = \begin{cases} 1 & \text{if } D_{ij} \equiv 0 \mod 3 \\ 0 & \text{otherwise} \end{cases}$$

and similarly for $R^{(1)}$ and $R^{(2)}$. This is exactly solving the 'medium' case above 3 times, so the runtime is a total of $O(n^\omega \log^2 n)$.

References

[AMS99] Noga Alon, Yossi Matias, Mario Szegedy. The Space Complexity of Approximating the Frequency Moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.

[MR95] Rajeev Motwani and Prabhakar Raghavan. 1995. Randomized Algorithms. Cambridge University Press, New York, NY, USA.