CS 354: Unfulfilled Algorithmic Fantasies June 5, 2019

Lecture 19: Sub-cubic equivalences

Lecturer: Aviad Rubinstein *Scribe: Nick Guo*

1 Introduction

In this lecture, we will go over problems that are conjectured to take $\Omega(n^3)$ time.

1.1 Definitions

Definition 1.1 (**All Pairs Shortest Paths** (APSP)).
Input: Fully connected, directed graph $G = (V, E)$ where $|V| = n$, with real valued edge weights.

Output: Matrix A where each entry i, j is the shortest path length from vertex i to vertex j.
Note that the Floyd-Warshall Algorithm solves APSP in $O(n^3)$.

Definition 1.2 (**Negative Weight Triangles** (NEG-∇)).
Input: Fully connected, directed graph $G = (V, E)$ where $|V| = n$, with integer valued, polynomially-bounded edge weights.

Output: Vertices i, j, k s.t. $w_{i,j} + w_{j,k} + w_{k,i} < 0$. Here, $w_{a,b}$ denotes the weight of edge $a \to b$.
A brute force algorithm that enumerates all triples runs in $O(n^3)$.

Definition 1.3 (**Radius** (RADIUS)).
Input: Fully connected, directed graph $G = (V, E)$ where $|V| = n$, with real valued edge weights.

Output: Both the distance and the vertex satisfying

$$\min_{u \in V} \max_{v \in V} \operatorname{dist}(u, v)$$

The vertex u that achieves the above can be thought of as the center of the graph. Note that this problem can easily be solved given a solution to APSP, so also in $O(n^3)$ time.

1.2 Motivating Question

We're interested in whether or not is it possible to do notably better than $O(n^3)$ for any of those problems. It's been shown that we can do it in $O(n^{3-\Omega(\frac{1}{\sqrt{\log n}})})$ [Wil18], but existence of an $O(n^{2.99})$ algorithm is an open problem. Although we don't know of any such algorithms, there are reasons to be hopeful: for example, some related problems like unweighted triangle detection admit non-trivial algorithms based on matrix multiplication.

For the time being, we can show the existence of truly-subcubic time algorithms[1] for the three

[1]By *truly-subcubic time* algorithms we mean algorithms that run in time $O(n^{3-\epsilon})$ for some constant $\epsilon > 0$.

problems is equivalent (i.e. if one of the problems has a truly-subcubic time algorithm, then all three do).

Theorem 1.4 ([WW18])**.** *Either or all or none of the following problems admit truly subcubic time algorithms:* APSP*,* NEG-∇*,* RADIUS*.*

2 APSP $\iff$ NEG-∇

We will prove the reduction between APSP and NEG-∇ by showing a sequence of equivalence results. In particular, we will define two intermediate problems: 2-HOP APSP where it's APSP except the path length uses exactly two edges, and ALL PAIRS NEG-∇ which outputs an $n \times n$ boolean matrix representing whether edge i, j belongs to a negative triangle. (2-HOP APSP and ALL PAIRS NEG-∇ aren't really canonical problems, but are just tools to more easily relate APSP and NEG-∇.)

2.1 APSP $\iff$ 2-HOP APSP

($\Longleftarrow$) Suppose we have a $\leq k$-HOP APSP algorithm[2]. We produce a $\leq k^2$-HOP APSP algorithm.

On input G, run the $\leq k$-HOP APSP algorithm to produce D, the all pairs shortest path matrix with path length at most k. We then use this output to produce a new graph G' where the edge weight $w'_{i,j}$ is exactly $D_{i,j}$. We run the $\leq k$-HOP APSP algorithm again on G', and this output computes APSP with at most k edges from G', each edge of which is comprised of at most k edges in the original G. Thus, we see that by running our $\leq k$-HOP APSP algorithm twice, we have a $\leq k^2$-HOP APSP algorithm.

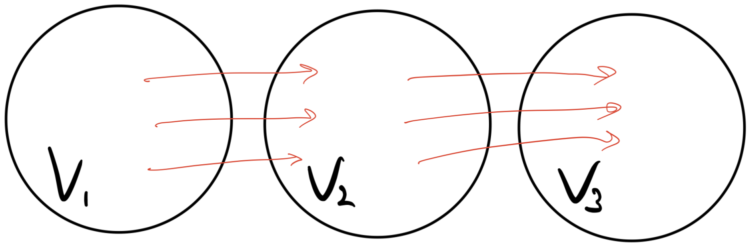
We can repeat this process $\log\log(n)$ times until we have an algorithm that's equivalent for APSP.

($\Longrightarrow$) Suppose we have an algorithm for APSP. We produce an algorithm for 2-HOP APSP.

On input $G = (V, E)$ with weights w, we will make a new graph G'. First, make 3 copies of V to get V_1, V_2, V_3. Within each copy, there are no edges. The only edges go from $V_1 \to V_2 \to V_3$. We define the transition function to go from vertex i in one set to vertex j in the next set as follows:

$$w'(v_l^i, v_{l+1}^j) := w_{i,j}.$$

[2]Here, similarly to 2-HOP APSP, $\leq k$-HOP APSP is the version where the path must use at most k edges. Note that an algorithm for 2-HOP APSP immediately implies an algorithm for $\leq$ 2-HOP APSP by taking the entry-wise minimum of the 2-HOP APSP and original edge-weight matrices.



We now run APSP on G', and we see that the resulting output was forced to go through at most 2 edges from the original G. For the purposes of these equivalences, $\leq$ 2-HOP APSP is sufficient.

2.2 2-HOP APSP $\iff$ ALL PAIRS NEG-∇

($\implies$) Suppose we have an 2-HOP APSP algorithm. We show an ALL PAIRS NEG-∇ algorithm.

On input G, run the 2-HOP APSP algorithm to get $\triangle^2$, the exactly two length path APSP matrix. Then, for every edge, we check whether it belongs to a negative triangle by taking

$$IsNeg(i,j) = (\triangle^2_{i,j} + w_{i,j}) < 0$$

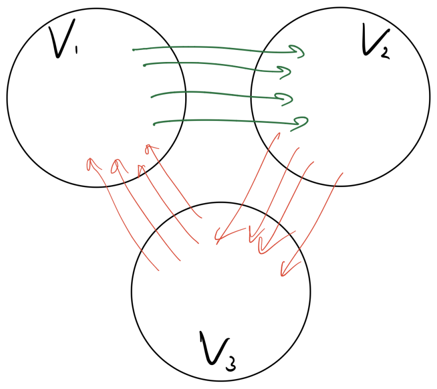
($\impliedby$) Suppose we have an ALL PAIRS NEG-∇ algorithm. We show an 2-HOP APSP algorithm.

This direction is a little more tricky than the others because we want to turn a binary signal into actual values for 2-HOP APSP.

Consider one edge $i \to j$. We use the ALL PAIRS NEG-∇ solver to compute the two-edge shortest path from i to j one bit at a time (binary search). We change the weight of edge $i \to j$ from most significant bit to least significant bit using the signal from ALL PAIRS NEG-∇ for i, j. For example, if i, j belongs to a negative triangle, then the value for the exact two path must be larger than the current weight. Note that this requires as many queries as the size of the value representation — here we use the assumption that the values are polynomially-bounded integers.

We can parallelize the above process to save a factor of $|V|^2$ queries to the ALL PAIRS NEG-∇ solver. To do this, make three copies of the vertices, V_1, V_2, V_3. We fully connect the vertices from set to set into triangles $V_1 \to V_2 \to V_3 \to V_1$, but within each vertex set, there are no edges. The weights are defined using the original weights

$$w'(v^i_l, v^j_{l+1 \pmod 3}) := w_{i,j}.$$



Then, we freeze the weights for $V_2 \to V_3$ and $V_3 \to V_1$, and run the parallel binary search by changing only the $V_1 \to V_2$ edge weights. Finally, we return the negative of the $V_1 \to V_2$ adjacency matrix as the 2-HOP APSP output.

2.3 All Pairs Neg-∇ $\iff$ Neg-∇

($\implies$) This direction is trivial since All Pairs Neg-∇ captures existence of a negative triangle.

($\impliedby$) Suppose we have a Neg-∇ algorithm. We show an All Pairs Neg-∇ algorithm.

This is the most interesting reduction because we're comparing a problem with 1 output to a problem with n^2 binary outputs. Thus, the intuition for getting sub-cubic runtime is that we need to be solving smaller problems.

Our initial setup will be the same as the ($\impliedby$) direction of 2.2, so we make 3 copies of the input vertices, V_1, V_2, V_3, and again there are no edges within each V_i. We will pass the Neg-∇ solver subproblems of $V' = A \cup B \cup C$ where $A \subset V_1, B \subset V_2, C \subset V_3$, $|A| = |B| = |C| = n^{1/3}$, and $A \cap B \cap C = \emptyset$ with respect to the original vertices.

Parameters and back-of-the-envelope calculation To create A, B, C, we partition each of V_1, V_2, V_3 into subsets of size $n^{1/3}$ so that there are a total of $n^{2/3}$ subsets for each. A subproblem consists of just a choice of one subset from each partition, so there are $n^{2/3} \cdot n^{2/3} \cdot n^{2/3} = n^2$ triplets (A, B, C). The size of each subproblem instance is $3 \cdot n^{1/3}$, so if our Neg-∇ algorithm runs in $O(n^{3-\delta})$, then solving each subproblem takes time $O(n^{1-\delta/3})$.

We proceed with the following algorithm.

```
Let $M$ be an $n \times n$ binary matrix which is zero-initialized.

For each subproblem $(A, B, C)$:
    While Neg-$\nabla$ on the current subproblem returns vertices $i \in A, j \in B, k \in C$:
      • Mark $M_{i,j} = 1$, indicating edge $i \to j$ belongs to a negative triangle
```

- Set weight of edge $i \to j$ to be ∞ for this and all later subproblems.
  Note: since $i \in A, j \in B, k \in C$, we are only changing weights for edges from $A \to B$.

Return M as the All Pairs Neg-∇ output.

Correctness Setting an edge weight to ∞ after discovering it in a negative triangle allows the Neg-∇ solver to find other negative triangles on later queries. Otherwise it's possible to keep receiving back the same negative triangle even in different subproblems, so this guarantees that we always make progress. Notice that we only set one entry of M to 1 upon discovery of a negative triangle even though there are three edges that should have their M entries set to 1. This is okay because those edges will come up in later subproblems since the corresponding $i \to j$ edge whose weight was set to ∞ is unchanged from $V_2 \to V_3$ and $V_3 \to V_1$.

Running time The while loop is necessary as multiple negative triangles may exist within one subproblem. Notice that we may have to call the Neg-∇ as many as $n^{2/3}$ times for each subproblem, which doesn't seem to match with our back-of-the-envelope calculation above (which assumed one call to Neg-∇ for each (A, B, C)). To resolve this, we upper bound the *total* number of times that we call Neg-∇. Notice that we only call Neg-∇ again for the same (A, B, C) after we mark $M_{i,j} = 1$ for some i, j. Furthermore, after we mark $M_{i,j} = 1$, we will never find another triangle using that edge (although we continue to find triangles using other copies of that edge). Therefore,

$$\begin{aligned}(\#\text{ of calls to Neg-}\nabla) &= (\#\text{ of triplets } (A, B, C)) + (\sum_{i,j} M_{i,j}) \\ &\le n^2 + n^2.\end{aligned}$$

Therefore, we make a total of $O(n^2)$ calls to the Neg-∇ solver across all subproblems. Hence, as in our back-of-the-envelope calculation, the total runtime of our algorithm (assuming $T^{\text{Neg-}\nabla}$ is $O(n^{3-\delta})$) is given by

$$O(n^2) \cdot T^{\text{Neg-}\nabla}(3n^{1/3}) = O(n^{3-\delta/3}).$$

3 Radius $\iff$ APSP, Neg-∇

($\implies$) As hinted earlier, this is trivial since we take the APSP output, and find the vertex satisfying the Radius property.

($\impliedby$) Suppose we have a Radius algorithm. We show a Neg-∇ algorithm.

On input graph $G = (V, E)$, we will clone the vertex set four times to get V_1, V_2, V_3, V_4. Again, within each V_i, there are no edges, and we fully connect $V_1 \to V_2 \to V_3 \to V_4$ using the original edge weights. Observe that for any vertex u, if we let u_1 be the instance of u in V_1, and u_4 be the instance in V_4, the smallest triangle around u is

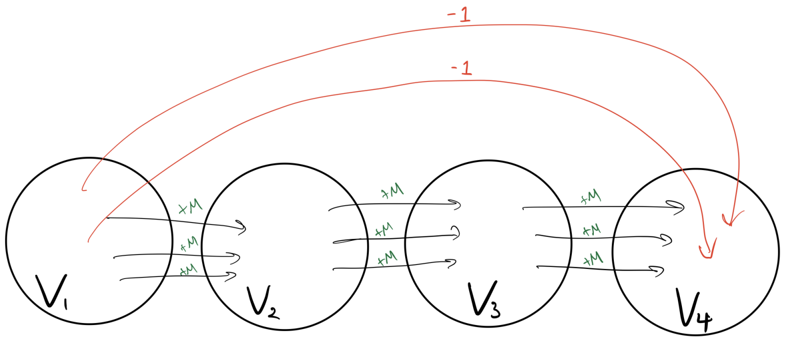
$$\triangle(u_1, u_4) = \min_{x \in V_2, y \in V_3} w(u_1, x) + w(x, y) + w(y, u_4)$$

This doesn't quite work because it could be the case that the largest distance occurs from V_1 to V_2 instead of extending all the way to V_4 (perhaps later edge weights are negative). We can just add a really large constant M to the edge weights of our new graph so that we expect any path from V_1 to V_4 to be at least $3M$ if there are no negative triangles, and this dominates one edge paths from V_1 to V_2.

Now we want to make sure that the output of Radius doesn't correspond to an open three-edge path (i.e. one that doesn't close a triangle). To account for this, we add shortcut edges from $V_1 \to V_4$. For $u_1 \in V_1, v_4 \in V_4$

$$w(u_1, v_4) = -1 \text{ if } u \neq v$$

Thus, any path that doesn't start and end at the same original vertex has a negative distance, so the inner max term will ignore them if all triangles containing that vertex are non-negative.



Finally, to detect a negative triangle, we can take the output of Radius and compare it to $3M$. If it's smaller, then there must have been a negative triangle. It's then a simple $O(n^2)$ scan starting from the center vertex to find the vertices of the negative triangle.

References

[Wil18] R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018.

[WW18] Virginia Vassilevska Williams and R. Ryan Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27:1–27:38, 2018.