1 Approximate Shortest Paths

In this lecture, we will show an additive +2-approximation for All-Pairs Shortest Paths (APSP) on undirected unweighted graphs. That is, for each pair of vertices $u, v \in V$ our algorithm will output an estimate $\tilde{d}(u,v)$ such that $d(u,v) \leq \tilde{d}(u,v) \leq d(u,v) + 2$.

Recall that Seidel's algorithm gives an exact algorithm for APSP in time $\tilde{O}(n^\omega)$. Here, we will give a +2-approximation for APSP in time $\tilde{O}(n^{7/3})$, which is better than $\tilde{O}(n^\omega)$ for the current best known bound on ω. As a warm-up, we will give an $\tilde{O}(n^{2.5})$ time +2-approximation algorithm.

Theorem 1.1. *There is an $O(n^{2.5}\sqrt{\log n})$ time algorithm that computes a +2-approximation to APSP.*

Proof. The pseudocode is given in Algorithm 1, but we also describe the algorithm here.

We will use the high degree-low degree technique and the hitting set technique. Let Δ be a parameter that we will set later and call a vertex *high-degree* if its degree is at least Δ and otherwise call it *low-degree*.

Compute a set S of size $O(\frac{n}{\Delta}\log n)$ that hits the neighborhood of every high-degree vertex. Recall that this set can be chosen randomly or we can use the greedy hitting set algorithm to choose the set deterministically. Perform BFS from each vertex in S. This takes time $O(\frac{n^3}{\Delta}\log n)$

Consider a pair u, v of vertices and consider a shortest path P between them. We will consider two cases: either P contains a high-degree vertex or every vertex on P is low-degree.

For the case where P contains a high-degree vertex x, we know that S hits a neighbor s_x of x. We claim that $d(u,s_x)+d(s_x,v)$ gives a +2-approximation for $d(u,v)$ (note that we can compute this quantity because we performed BFS from s_x). First, by the triangle inequality, $d(u,v) \leq d(u,s_x) + d(s_x,v)$. Again, by the triangle inequality, $d(u,s_x) \leq d(u,x)+1$ and $d(s_x,v) \leq d(x,v)+1$, so $d(u,s_x)+d(s_x,v) \leq d(u,v)+2$.

For the case where every vertex on P is low-degree, we let G_{low} be the subgraph of G that contains the set of edges incident to at least one low-degree vertex. We perform APSP on G_{low}. The path P is in G_{low} so this finds the exact value $d(u,v)$. Also note that for pairs of vertices from the previous case, this APSP overestimates their true distance. The number of edges in G_{low} is at most $n\Delta$ so this step takes time $O(\Delta n^2)$.

The running time of the entire algorithm is $O(\frac{n^3}{\Delta}\log n + \Delta n^2)$. Optimizing for Δ, we get that $\Delta = \sqrt{n \log n}$ so the running time is $\tilde{O}(n^{2.5}\sqrt{\log n})$.

Algorithm 1: +2-Approx APSP in time $\tilde{O}(n^{2.5})$

```
S ← a set of size O((n/Δ) log n) that hits the neighborhood of every high-degree vertex
foreach s ∈ S do
  run BFS from s
foreach u, v ∈ V do
  d_1(u,v) ← min_{s∈S}(d(u,s) + d(s,v))
G_low ← the subgraph of G that contains the set of edges incident to at least one low-degree vertex
Run APSP in G_low and let d_2(u,v) be the distances found
foreach u, v ∈ V do
  return min{d_1(u,v), d_2(u,v)}
```

□

Theorem 1.2 ([1]). *There is an $\tilde{O}(n^{7/3})$ time algorithm that computes a +2-approximation to APSP.*

Proof. The pseudocode is given in Algorithm 2, but we also describe the algorithm here.

The idea of this proof is that instead of just having high-degree and low-degree vertices, we will also have *medium-degree* vertices. Let R and Δ be parameters to be set later with $R < \Delta$. We say that a vertex is high-degree if its degree is at least Δ, medium-degree if its degree is less than Δ and at least R, and low-degree if its degree is less than R.

Again, we choose a set S of size $O(\frac{n}{\Delta}\log n)$ that hits the neighborhood of each high-degree vertex. Additionally, we choose a set T of size $O(\frac{n}{R}\log n)$ that hits the neighborhood of each medium-degree vertex.

Like before, we perform BFS from each vertex in S, which handles all shortest paths that contain a high-degree vertex. This takes time $O(\frac{n^3}{\Delta}\log n)$.

Now, we define the graph G_{med} to be the subgraph of G that contains the set of edges incident to at least one low or medium-degree vertex. Instead of computing APSP on this graph like we did for G_{low} in the previous algorithm, we only compute BFS in G_{med} from the vertices in our sample T. The number of edges in G_{med} is at most $n\Delta$ so this step takes time $\tilde{O}(\frac{n}{R}\Delta n)$.

Now comes the most interesting and clever part of the algorithm. For every vertex v we create a new graph G_v. G_v is a weighted graph on the same vertex set as the original graph with the following edges:

- Include all edges incident to at least one low-degree vertex. This is at most nR edges.
- For each medium-degree vertex x, let $t_x \in T$ be a neighbor of x and include the edge (x, t_x). Recall that such a t_x exists by choice of T. This is at most n edges.
- For each vertex $t \in T$, include a new edge from t to v whose weight is the distance in G_{med} between t and v. Recall that we calculated this distance when we computed BFS in G_{med} from the vertices of T. This is at most $|T| = O(\frac{n}{R}\log n)$ edges.

Then, for every vertex v we run Dijkstra's algorithm from v in the graph G_v. Each G_v contains $\tilde{O}(nR)$ edges so this takes time $\tilde{O}(n^2R)$. We will prove that this algorithm is correct after calculating the running time.

The running time of the entire algorithm is $\tilde{O}(\frac{n^3}{\Delta} + \frac{n^2}{R}\Delta + n^2R)$. Setting the first two terms equal we get that $\Delta = \sqrt{Rn}$. Setting the first and third term equal we get that $R = n^{1/3}$. Thus, $\Delta = n^{2/3}$. Therefore, the total running time is $\tilde{O}(n^{7/3})$.

Correctness We have already shown correctness for pairs u, v of vertices whose shortest path P contains a high-degree vertex. So, suppose P contains no high-degree vertices. Recall that we performed Dijkstra's algorithm from u in G_u. First, note that distances in G_u cannot underestimate distances in the original graph, so our algorithm will never always return an estimate that is at least $d(u, v)$. G_u contains all edges incident to low degree vertices so if P only contains low-degree vertices then we have found an exact shortest uv-path.

Thus, suppose P contains at least one medium-degree vertex. Let x be the last medium-degree node on P (i.e. the farthest from u). Recall that G_u contains an edge from x to t_x. Further recall that G_u contains a weighted edge from u to t_x. The entire path P is contained in G_{med} since we assumed that P has no high-degree vertices. Also, since x is of medium degree, the edge (x, t_x) is also in G_{med}. Thus, by the triangle inequality, the distance in G_{med} from u to t_x is at most $d(u, x) + 1$. So, the edge in G_u from u to t_x has weight at most $d(u, x) + 1$.

We can form a uv-path in G_u by taking the edge from u to t_x followed by the edge from t_x to x followed by the subpath of P from x to v. Note that this entire subpath is indeed in G_u since all vertices after x on P are of low degree. Therefore, by the triangle inequality, the distance between u and v in G_u is at most $d(u, x) + 1 + 1 + d(x, v) = d(u, v) + 2$.

Algorithm 2: +2-Approx APSP in time $\tilde{O}(n^{7/3})$

$S \leftarrow$ a set of size $O(\frac{n}{\Delta}\log n)$ that hits the neighborhood of every high-degree vertex
$T \leftarrow$ a set of size $O(\frac{n}{R}\log n)$ that hits the neighborhood of every medium-degree vertex
foreach $s \in S$ **do**
  run BFS from s to compute $d(s,v)$ for all $v \in V$
foreach $u, v \in V$ **do**
  $d_1(u,v) \leftarrow \min_{s\in S}(d(u,s)+d(s,v))$
$G_{med} \leftarrow$ the subgraph of G that contains the set of edges incident to at least one low or medium-degree vertex
foreach $t \in T$ **do**
  run BFS in G_{med} from t to compute $d_{\text{med}}(t,v)$ for all $v \in V$
foreach $u \in V$ **do**
  initialize a new graph $G_u = (V, E_u)$ where E_u is initialized to $\emptyset$
  add to E_u every edge incident to at least one low-degree vertex
  foreach *medium-degree vertex* x **do**
    $t_x \leftarrow$ an arbitrary vertex in $N(x) \cap T$
    add (x, t_x) to E_u
  foreach $t \in T$ **do**
    add to E_u the edge (u,t) with weight $d_{med}(u,t)$
  run Dijkstra's algorithm from u in G_u to obtain distances $d_2(u,v)$ for all $v \in V$
foreach $u, v \in V$ **do**
  return $\min\{d_1(u,v), d_2(u,v)\}$

□

After seeing these two algorithms, you might wonder whether we can get a better algorithm by partitioning the vertices into more than 3 sets based on degree. In fact, there are algorithms that do this, however they get worse approximation factors. For example, on the other side of the time/accuracy trade-off there is an algorithm that runs in time $\tilde{O}(n^2)$ and achieves a $+\log n$-approximation. The $\tilde{O}(n^{7/3})$ time algorithm remains is the fastest known algorithm for getting a +2-approximation.

References

[1] Dor, D., Halperin, S., and Zwick, U. (2000). All-pairs almost shortest paths. *SIAM J. Comput.*, 29(5), 1740-1759.