Advanced Algorithms 2021 11/19, 2021

Exercise 09

Lecturer: Mohsen Ghaffari *Teaching Assistant: Michal Dory*

1 Basic HopSets [Recommended]

Consider an arbitrary weighted n-node graph $G = (V, E, w)$ where $w : E \to \mathbb{R}^+$ indicates the weight of the edges. For two nodes $s, t \in V$, let $dist(s, t)$ denote the smallest total weight among all paths that connect s and t. Moreover, define the *shortest-path hop-distance* of s and t to be the smallest h such that there is a path with weight $dist(s, t)$ and only h hops, connecting s and t. We define the Shortest-Path Hop-Diameter (SPHD) of the graph to be the maximum shortest-path hop-distance for any pair of vertices s and t.

In general, a graph might have a large SPHD, even if all nodes are within few hops of each other (think of a wheel graph with heavy weights on the spokes and light weights around the wheel). Working with graphs of smaller SPHD is much easier, in many computational settings. Prove that if for each node v, we add weighted edges to its closest k nodes (breaking ties arbitrarily) with weight equal to the shortest path between v and that node, we obtain a graph that preserves pair-wise distances, while having SPHD at most $4n/k$.

Hint:
Suppose an optimal path in the new graph and observe that there is no new edge between any pair of non-consecutive nodes of this path.

2 A Near-Linear Time Algorithm for Multiplicative Spanner [Recommended]

In the class, we saw that a simple greedy method yields a $(2k-1)$-multiplicative spanner with $O(n^{1+1/k})$ edges, and the algorithm can be performed in polynomial time. In this exercise, we see an algorithm that achieves essentially the same objective in near-linear time, that is, in $\tilde{O}(m)$ time, where m denotes the number of the edges in the graph.

The algorithm has $k-1$ phases main phases, and works with a rooted clustering of vertices. Initially, each vertex forms its own cluster. In each phase, we do as follows: first, for each cluster, we add a BFS-tree confined to this cluster and rooted at the root of this cluster, to the spanner. Then, we mark each cluster-root as surviving with probability $n^{-1/k}$ and dead otherwise. The corresponding cluster is called similarly.

Then, each vertex v that was in a dead cluster panics and looks for a new cluster, as follows: If v has a neighbor u that is in a surviving cluster of this phase, then v picks one of those neighbors u and joins the same cluster as u. If v has no such neighbor u, then v becomes hopeless and gives up on the clustering process: in that case, for each (dead) cluster C that has a node u neighboring v, node v adds one edge connecting v to some node in C to the spanner. From this point on, node v does not participate in the algorithm anymore.

Then, the algorithm has one final phase, where here each node behaves as a node in a dead cluster, and adds one edge to each of its neighboring clusters.

(A) Prove that the expected number of edges added to the spanner in the above scheme is at most $O(kn^{1+1/k})$ [1].

[1] With some more care in the analysis, you can tighten the bound to $O(n^{1+1/k} + nk)$ edges.

(B) Prove that, we always get a $(2k-1)$-multiplicative spanner.

(C) Explain how the above algorithm can be performed in $\tilde{O}(m)$ time.

Hint:

After phase i, the diameter of each cluster is at most $2i$.

3 Additive Approximation of All Pairs Shortest Path

Devise an algorithm with complexity $\tilde{O}(n^{5/2})$ that computes a 2-additive approximation of all pair shortest paths. That is, for every pair of vertices s, t, the algorithm should output a value that is in $[dist(s,t), dist(s,t)+2]$.

Hint:

Find all the distances on a 2-additive spanner of the original graph.

4 6-Additive Spanner

In this exercise, we prove theorem 6 of the lecture notes, by showing that each n-node graph has a 6-additive spanner with $\tilde{O}(n^{4/3})$ edges.

(A) As usual, the algorithm first easily takes care of *low-degree* vertices, those of degree at most D_1, by adding all of their edges. How large can we set D_1?

(B) The second usual step in the algorithm is to take care of shortest paths that have at least one high-degree vertex, by sampling a few nodes at random, and adding one BFS per sampled node. We call a vertex *high-degree* if its degree is above D_2. How small can we set D_2?

Now we remain with shortest paths that have no high-degree vertex, and for each such shortest path, the edges incident on low-degree vertices are already purchased. So, we only need to pay for the extra edges that we add. We handle these paths in $O(\log n)$ successive iterations, where in the i^{th} iteration we handle paths for which roughly 2^{i-1} edges are not purchased before. In particular, for each $i \in \{0, 1, \dots, 2\log n\}$, define set T_i by randomly including each node $v \in V$ in T_i with probability $\frac{C \log n}{D_1 2^i}$, for some sufficiently large constant C. For each $i \in \{0, 1, \dots, 2\log n\}$, and for each $t_0 \in T_0$ and $t_i \in T_i$, let P_i be the shortest path between t_0 and t_i that has at most 2^{i-1} edges that remain (not already in H) and add all of its edges to the spanner H.

For the last step of the construction, for each node that has a neighbor in T_0, we add an edge to one such neighbor in the spanner.

(C) Prove that the expected number of edges in the above scheme is $\tilde{O}(n^{4/3})$.

(D) Prove that, with high probability, we get a 6-additive spanner.