University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
P. Schneider

Advanced Algorithms
Problem Set 8

Issued: Friday, June 28, 2019

Exercise 1: Almost Linear-Time Multiplicative Spanner Algorithm

In the lecture, we have seen an algorithm that computes a $(2k$–$1)$-multiplicative spanner with $O(n^{1+1/k})$ edges of a given n-node graph $G = (V, E)$ in time polynomial in n. In this exercise, we will analyze a randomized algorithm that allows to compute a multiplicative spanner with almost the same guarantees. However, the algorithm has a very efficient distributed implementation and it can also be implemented in time $\tilde{O}(m + n)$[1] sequentially (where $m = |E|$).
The algorithm has a parameter $k \geq 1$ and it runs in k phases. Throughout the k phases, the set of nodes are partitioned into active and inactive nodes and the active nodes are partitioned into clusters. The algorithm also maintains a set $E_S \subseteq E$ of edges to be added to the spanner. Initially, $E_S = \emptyset$, all nodes are active, and each node forms a cluster by itself. For ease of description, assume that each node $v \in V$ has a unique identifier $\mathrm{ID}(v)$ and also that each cluster C has a unique identifier $\mathrm{ID}(C)$ (initially, the cluster IDs of the single node clusters are equal to the IDs of their nodes). In the following, we describe how the set E_S, the set of active and passive nodes, and the clusters are updated in each phase $i = 1, \ldots, k$.

1. If $i \leq k - 1$, set $p := n^{-1/k}$, otherwise set $p := 0$. For each cluster C, independently mark C with probability p. At the end of the phase, only the marked clusters will survive to the next phase.
2. For each node $v \in V$ in an unmarked cluster, do the following.
   (i) If v has some neighbor $u \in V$ that is in a marked cluster C, add *one* such edge $\{v, u\}$ to E_S. At the end of the phase, v joins cluster C.
   (ii) If v has no neighbor in a marked cluster, for each cluster C' in which v has a neighbor, v adds *one* edge $\{v, u\}$ to some neighbor $u \in C'$. At the end of the phase, v becomes inactive. Additionally, v is not in a cluster any more.

Finally, the algorithm outputs the graph induced by the edge set E_S as the spanner.

(a) Show that for each $i < k$, at the end of phase i, the set of spanner edges E_S contains a spanning tree of depth at most i for each of the remaining clusters.

Note that this implies that for each edge $\{u, v\} \in E$ between two nodes in the same cluster, the spanner contains a path of length at most $2i$.

(b) Show that for each node $u \in V$ that gets deactivated in phase $i \leq k$, for each neighbor v of u, at the end of the phase, the spanner contains a path of length at most $2i - 1$ between u and v. Argue why this implies that the multiplicative stretch of the spanner is at most $2k - 1$.

(c) Show that for $k = O(\log n)$, the spanner at the end with high probability contains at most $O(n^{1+1/k} \log n)$ edges.

(d) Sketch how (for $k = O(\log n)$), the algorithm can be implemented in $\tilde{O}(m + n)$ time (where $m = |E|$).

[1] Recall that the $\tilde{O}(\cdot)$-notation hides polylogarithmic factors, i.e., $\tilde{O}(f(n)) = f(n) \cdot (\log f(n))^{O(1)}$.

Exercise 2: Multiplicative Spanners in Weighted Graphs

Let $G = (V, E, w)$ be a graph with edge weights $w(e) > 0$. The notion of an α-multiplicative spanner can naturally be extended to weighted graphs: For every two nodes $u, v \in V$, the spanner needs to contain a path of weighted length within an α-factor of the (weighted) distance between u and v in G. Describe how the $(2k-1)$-multiplicative spanner algorithm from the lecture can be adapted to weighted graphs so that it still only requires $O(n^{1+1/k})$ edges.

Do you also see how the randomized algorithm of Exercise 1 can be adapted to weighted graphs? (Note that this is much less straightforward than adapting the algorithm from the lecture.)

Exercise 3: Additive Approximation of All Distances in a Graph

Devise an algorithm with running time $\tilde{O}(n^{5/2})$ that computes a 2-additive approximation of all distances of an unweighted n-node graph $G = (V, E)$. That is, the algorithm should output a value $\hat{d}(u, v) \in [d_G(u, v), d_G(u, v) + 2]$ for all pairs of nodes $u, v \in V$.