Advanced Algorithms 2024 11.11, 2024

Exercise 08

Lecturer: Maximilian Probst *Teaching Assistant: Patryk Morawski*

Terminology:

1. Recall that $\tilde{\mathcal{O}}(f(n)) = \mathcal{O}(f(n)(\log n)^c)$, where c is some constant.
2. We say that a graph S is a k-spanner for G if if $dist_G(u,v) \leq dist_S(u,v) \leq k \cdot dist_S(u,v)$ for all $u, v \in V(G)$.

1 A Near-Linear Time Algorithm for Spanners

In the lecture you have seen a polynomial-time algorithm that given any graph G computes a $(2k-1)$-spanner for G with $\mathcal{O}(n^{1+1/k})$ edges, which runs in polynomial time. In this exercise, we want to analyze an algorithm that will perform basically the same task in almost linear time, i.e., in time $\tilde{\mathcal{O}}(m)$, where m is the number of edges of G.

The algorithm will consist of $k-1$ main phases, in each of them we will add some new edges to our spanner H, which is empty in the beginning. During the algorithm, we will maintain a set of clusters (disjoint sets of vertices) that will change throughout. In the beginning, every vertex forms its own cluster.

Then, in each phase we do as follows. For each of the existing clusters, we mark it as surviving with probability $n^{-1/k}$, independently from all the other clusters. Otherwise, we say that this cluster is dead. Each vertex v that is in a dead cluster then panics and looks for another cluster it can join. If v has a neighbor u that is in a surviving cluster, then v picks one of these neighbors arbitrarily and joins the cluster of u. We add the edge uv to our spanner H. Otherwise, if v doesn't have such a neighbor, for each cluster C that contains some neighbor of v, we add an edge connecting v with some vertex of C to our H. From this point on, v won't participate in the algorithm anymore.

After $k-1$ of such phases, we perform one final step, where each vertex behaves as a vertex in a dead cluster: for each remaining vertex v, we add one edge to each of the neighboring clusters of v.

1. Prove that the expected number of edges added to H in this process is at most $\mathcal{O}(kn^{1+1/k})$.
2. Prove that at the end of this process H is a $(2k-1)$-spanner for G.
3. Argue that the above algorithm can be performed in $\tilde{\mathcal{O}}(m)$ time. Even though it's not necessary, you can assume that $k = \mathcal{O}(\log n)$.

2 Very Sparse Spanners

In the lecture, you have seen an algorithm that utilizes expander decomposition to obtain a cut-preserving sparsifier with $\tilde{\mathcal{O}}(n)$ edges. In this exercise, we want to show that the same construction also gives us a $\tilde{\mathcal{O}}(1)$[1]-spanner for any **unweighted** graph G.

1. Let H be a ϕ-expander, where $\phi = \tilde{\Omega}(1)$, and let $u, v \in V(H)$ be arbitrary vertices. Suppose that H has no isolated vertices. Show that $dist_H(u,v) = \tilde{\mathcal{O}}(1)$.

[1]Recall that there are possible log factors hidded behind the $\tilde{\mathcal{O}}$ notation.

2. Let G be a graph with m edges and let $G_1, \ldots, G_k$ be the edge-decomposition of G into expanders given by Theorem 8.12. Prove that each G_i' obtained by running the down-sampling procedure on G_i (Algorithm 24) and removing the edge-weights (i.e. setting them all to one) is a $\tilde{\Omega}(1)$-expander with high probability.

3. Show that the graph $H = \cup_i G_i'$ is a $\tilde{\mathcal{O}}(1)$-spanner.

3 Cut-Preserving Sparsifiers for Graphs with Large Min-Cut

Let G be an unweighted graph with m edges and suppose that the size of a minimum cut in G is at least $\mu(G) = k$. Devise a randomized algorithm that for any given $\epsilon > 0$ computes a weighted graph H such that with high probability the following holds: H has at most $20 \cdot \frac{\log n}{\epsilon^2 k} m$ edges and for any $S \subseteq V(G)$ we have

$$(1-\epsilon)|E_G(S, V(G) \setminus S)| \leq |E_H(S, V(G) \setminus S)| \leq (1+\epsilon)|E_G(S, V(G) \setminus S)|.$$

Your algorithm should run in polynomial time. You can use the fact that for any $\alpha \geq 1$ the number of cuts of size at most αk in G is at most $(2n)^{\lceil \alpha n \rceil}$.