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

Advanced Algorithms
Sample Solution 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 $\text{ID}(v)$ and also that each cluster C has a unique identifier $\text{ID}(C)$ (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 $i = 1, \dots, k$.

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.

Remark: 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., $\tilde{O}(f(n)) = f(n) \cdot (\log f(n))^{O(1)}$.

Sample Solution

(a) Let C_i be some cluster that survived until phase $i < k$. We prove the claim by induction. Initially, cluster C_0 consists only of v, which forms the root. Presume that C_{i-1} is a tree of depth at most $i-1$ of the graph induced by the nodes in C_{i-1} and is marked again by the algorithm. Then C_i is the graph induced by C_{i-1} plus all edges with one endpoint in C_{i-1} and another in an unmarked cluster.

We connect each new node in C_i with exactly one edge to some node in C_{i-1}. The result is again a spanning tree, since C_{i-1} was a spanning tree, so is C_i. Clearly the depth increases by at most one, which is why it is at most i.

(b) In the case that neighbor v of u was in the same cluster as u, we have that u and v are connected by a path of length $2(i-1)$ due to part (a). In the case that v and u were in different clusters and v, then u adds an edge to some node v' in the cluster C' of v. The distance between v and v' is at most $2(i-1)$ again due to part (a), hence there is a path of length at most $2i-1$ from u to v. If v is not in a cluster anymore, then v already added an edge to u before and the claim is true due to an inductive argument.

In the final phase k no cluster is marked anymore and each node executes step 2. (ii). That means each node is now connected to each of its neighbors by a path of length at most $2k-1$ in E_S as we showed before. Since the distance between *neighbors* increases by a multiplicative factor of at most $2k-1$ in the graph $G[E_S]$ induced by E_S, clearly the distance between two nodes G increases by at most the same factor when compared to the respective distance in $G[E_S]$.

(c) We are done if we can show that each node adds at most $O(kn^{1/k})$ edges, which we will do in the following. If v is marked or joins some cluster (steps 1. and 2. (i)) in a phase we add at most 1 edge in that phase. The total number of edges added per node in steps 1. and 2. (i) is $O(k)$.

If in phase $i < k$ some node v has at least $kn^{1/k}$ incident clusters (i.e. v's neighbors belong to at least that number of different clusters), then we show that one of these clusters is marked w.h.p., hence v executes step 1. or 2. (i) in phase i and adds only 1 edge w.h.p. Let $N \geq kn^{1/k}$ be the number of incident clusters of v. Then the probability that no incident cluster is marked is

$$(1-p)^N = \Big(1 - \frac{1}{n^{1/k}}\Big)^N \leq \Big(1 - \frac{1}{n^{1/k}}\Big)^{kn^{1/k}} \leq e^{-k} \stackrel{k:=c\ln n}{=} \frac{1}{n^c}.$$

If a node v is finally deactivated in step 2. (ii), then this can (w.h.p.) be only due to two reasons:

(I) It is $i < k$ and the number of clusters incident to v is strictly less than $kn^{1/k}$ w.h.p.

(II) It is $i = k$, then the overall number of clusters is $O(kn^{1/k})$ w.h.p. (to be shown below).

In case (I) we add at most $kn^{1/k}$ edges since that is the number of clusters v connects to. In case (II) we add $O(kn^{1/k})$ edges, because that is the overall number of cluster v can possibly connect to. This remains to be shown. The probability that a given cluster C survives $k-1$ phases is $p^{k-1} = 1/n^{\frac{k-1}{k}}$. Hence the expected number of surviving clusters of the initial n clusters is $n^{1/k}$. Let S be the set of surviving clusters. With a Chernoff bound[2] we get

$$\mathbb{P}\Big(|S| > (1+k)n^{1/k}\Big) \leq \exp\Big(-\frac{kn^{1/k}}{3}\Big) = \Big(\frac{1}{n^c}\Big)^{\frac{\sqrt[k]{n}}{3}} \leq \frac{1}{n^{c/3}}.$$

We union bound the event $|S| > (1+k)n^{1/k}$ together with the events from further above that nodes with at least $kn^{1/k}$ incident clusters have no *marked* incident cluster. For completeness we give the generic union bound for events that occur w.h.p. (c.f. Exercise Sheet 1 for the proof).

Lemma 1. *Let $E_1, \ldots, E_k$ be events each taking place w.h.p. If $k \leq p(n)$ for a polynomial p then $E := \bigcap_{i=1}^k E_i$ also takes place w.h.p.*

[2] We use the following Chernoff bound $\mathbb{P}\big(X > (1+\delta)\mu_H\big) \leq \exp\big(-\frac{\delta\mu_H}{3}\big)$, with $X = \sum_{i=1}^n X_i$ for i.i.d. random variables $X_i \in \{0,1\}$ and $\mathbb{E}(X) \leq \mu_H$ and $\delta \geq 1$.

(d) The state of a node v is given by the following information:

- Whether v is active or inactive (since round i),
- v's cluster is marked in round i,
- v's cluster ID.

Each phase, we need to efficiently update the status of each node, and also add the necessary edges to the spanner E_S along the way. Since the number of phases is in $\tilde{O}(1)$ we are allowed to iterate over V and E a constant number of times each phase.

Let the current round be $i < k$. First, we can implement the marking process of clusters (and relaying that information to all nodes of a cluster) as follows. The algorithm keeps a list of cluster roots (initially all nodes). Before each phase we iterate this list, and mark the root with probability p, and store the marked roots in a new, separate list for the next phase.

When a root is marked, we broadcast all nodes of that cluster by simply following the edges E_S starting from the root and updating the status of all nodes of that cluster we find to "marked in round i". The broadcast stops at nodes that are not part of that cluster. Since the clusters are node-disjoint, edges within a cluster will be touched at most once. Edges spanning two different clusters will be touched at most twice. Thus all broadcasts can be conducted in $O(m)$.

Next we iterate V, and set all nodes that are still active (tentatively) to "inactive in round i". Then we iterate the set of edges E. If both endpoints of an edge belong to the same cluster or one endpoint is inactive since phase $i-1$ or longer, we do nothing. If one endpoint u is marked in round i and the other endpoint v is *not* marked in round i and also inactive in round i, then we add the edge to E_S and set v's cluster ID to u's (but we do not mark v) and set v to active again.

Finally, we iterate over the nodes once more. When we encounter a node v that still has the "inactive since round i" label, we iterate over the incident edges, and add the edge to E_S if v is not connected to some node in that cluster yet (we can keep track of the clusters v is connected to in $O(\deg(v))$). In this process we will touch each edge at most twice (once from each endpoint) which costs $O(m)$. A (high level) pseudo-code description of the above (for phase i):

Algorithm 1 LinearTimeSpannerPhase(G, k, i, L) ▷ *phase i, rootlist L, global variable E_S*

```
if i > k then return
else if i = k then p ← 0
else p ← n^{-1/k}
L' ← ∅
for each r ∈ L do
    if coinflip with probability p is successful then
        broadcast label "marked in round i" to cluster, stop at nodes not in cluster of r
        L' ← L' ∪ {r}
for v ∈ V do
    add label "inactive in round i" to v
for {u, v} ∈ E do
    if u is marked in round i and v is not marked in round i and v is inactive in round i then
        v gets cluster ID of u
        remove label "inactive in round i" from v
        E_S ← E_S ∪ {u, v}
for v ∈ V do
    if v is inactive in round i then
        for each cluster C in which v has a neighbor do
            v adds one edge {v, u} to E_S for some neighbor u ∈ C
LinearTimeSpannerPhase(G, k, i+1, L')
```

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.)

Sample Solution

We give a simple adaption of the greedy algorithm from the lecture to compute a $(2k-1)$-spanner with $O(n^{1+1/k})$ edges in the weighted case.

Algorithm 2 GreedySpannerWeighted$(G, w : E \to \mathbb{R}^+)$

```
E' ← ∅
for e = {u, v} ∈ E in ascending order by weight do
    if d_G'(u, v) > (2k−1) · w(e) then
        E' ← E' ∪ {e}
return G' = (V, E')
```

We can formulate the proof of correctness almost analogously. First of all it is clear that the stretch is at most $2k-1$ by construction of the algorithm. We prove that G' has girth $g(G') \geq 2k+1$. Assume that during the construction, we add an edge $e = \{u, v\}$ that closes a cycle with at most $2k$ edges (including e). Since we added edges in ascending order by weight, all edges on the cycle except e have weight at most $w(e)$. Thus $d_{G'}(u, v) \leq (2k-1) \cdot w(e)$. But then we would not have added e, a contradiction.

Remark: The adaptation of the algorithm in Exercise 1 to weighted graphs is the algorithm given in the seminal paper by Baswana and Sen "A simple and linear time randomized algorithm for computing sparse spanners" ICALP'03.

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 $u, v \in V$.

Sample Solution

We use the algorithm from the lecture to compute an additive 2-spanner E_S of G with $\tilde{O}(n^{3/2})$ edges in the same time. Then we conduct a BFS from every node on the spanner which takes time $O(n \cdot (n + |E|)) \subseteq \tilde{O}(n^{5/2})$.