ORIE 6334 Spectral Graph Theory October 6, 2016

Lecture 14

Lecturer: David P. Williamson *Scribe: Mika Sumida*

In this lecture, we show how to construct low-stretch trees. This is a bit out of order because we haven't yet said why having low-stretch trees is useful for anything having to do with spectral graph theory. But because fall break is happening next week, and the next few lectures build on each other, it's useful for us to explain how to construct such trees in this lecture, and all you'll have to remember after break is that we can do it.

1 Low-Stretch Trees

Definition 1 *Let $G = (V, E)$ be an undirected graph with weights $w(i, j) \geq 0$.*

*For a spanning tree T of G, the **stretch** of an edge (k, l) is*

$$st_T(k, l) = \frac{d_T(k, l)}{w(k.l)}$$

where $d_T(k, l) =$ sum of weights on k-l path in T.

*The **total stretch of T** is*

$$st_T(G) = \sum_{e \in E} st_T(e)$$

*The **average stretch of T** is*

$$\frac{1}{|E|} st_T(G)$$

Alon, Karp, Peleg, and West (1995) introduced the idea of low-stretch trees, and showed how to find a tree T with average stretch
$\leq \exp(O(\sqrt{\log n \log \log n}))$. Today we will look at a simplified version of their result for unweighted graphs.

2 Algorithm for Unweighted Graphs

Idea: We will partition G into low-diameter clusters, find trees in these clusters, shrink the clusters to vertices, and recurse.

[0]This lecture is derived from a lecture of Nick Harvey at the Sixth Cargèse Workshop on Combinatorial Optimization: `http://www.cs.ubc.ca/~nickhar/Cargese1.pdf`.

To deal with the recursion, it will be useful to consider multigraphs. Let $c(e)$ denote the number of copies of edge e in the multigraph. For a subset of edges $F \subset E$, let $c(F) = \sum_{e \in F} c(e)$. In the next section, we will describe a clustering algorithm that proves the following Lemma. The algorithm uses a standard technique called *region-growing*.

Lemma 1 *Let $C = c(E)$ and $D(C)$ be some parameter. Then there is a partition of G into clusters s.t.*

- *every cluster has diameter $\leq D(C)$ and*
- *there are at most $\alpha(C)|E|$ intercluster edges where $\alpha(C) \leq \dfrac{4\ln(C)}{D(C)}$*

*where e is an **internal edge** if both endpoints are inside the same cluster and is an **intercluster edge** otherwise.*

We will prove that our clustering algorithm satisfies this lemma but for now, assume that the lemma is true. Then the algorithm formalizing our idea is as follows:

Algorithm 1: LowStretchTree

Find partition of G into clusters $U_1, U_2, \cdots$, each with diameter $\leq D(C)$
foreach *cluster U_i* **do**
    Construct a shortest path tree T_i in U_i
Construct G' by contracting each U_i into a vertex u_i
$T' \leftarrow LowStretchTree(G')$
return $T' \bigcup_i T_i$

Theorem 2 (Alon et al. '95) *Pick any $\epsilon > 0$. Then the above algorithm returns a tree of average stretch $n^{O(\epsilon)}$.*

Proof: Let $\hat{T}$ be the tree returned by the algorithm. Pick a random edge $(i, j) \in E$ and let P be the i-j path $\hat{T}$. Let $U_1, U_2, \cdots$ be the clusters found in the clustering algorithm and let T' be the result of the recursive call.

If (i, j) is an internal edge in cluster U_k, then $d_{\hat{T}}(i, j) \leq 2D(C)$, since the diameter of U_k is at most $D(C)$.

If (i, j) is an edge between clusters U_k and U_l, let P' be the path in T' between U_k and U_l. Since (i, j) was chosen at random, its expected stretch is at most the average stretch of T'. Therefore,

$$\begin{aligned}
\text{expected \# edges on path } P &\leq (\text{expected \# edges in } P') \\
&\quad + (\text{expected \# vertices in } P')(\max_i diam(T_i)) \\
&\leq (\text{avg stretch of T'}) + (\text{avg stretch of T'} + 1)(2D(C)) \\
&\leq (\text{avg stretch of T'}) \cdot 5D(C)
\end{aligned}$$

Let $c'(e)$ be multiplicity of resulting edge e in G' and $c'(E) = C'$, and let $S(C) =$ worst case average stretch over all graphs G s.t. $c(E) = C$.

By Lemma 3, the maximum number of intercluster edges is $\alpha(C)|E|$ where $\alpha(C) = \frac{4\ln(C)}{D(C)}$, which implies that $C' \leq \alpha(C) \cdot C$.

Then

$$\begin{aligned} S(C) &\leq \underbrace{2D(C)}_{\text{stretch of internal edges}} + \underbrace{\alpha(C)}_{\text{fraction of intercluster edges}} \cdot \underbrace{S(\alpha(C)C) \cdot 5D(C)}_{\text{stretch of intercluster edge}} \\ &\leq 2D(C) + 20\ln(C)S\big(\alpha(C)C\big) \end{aligned}$$

If we set $D(C) = 4\ln(C)C^{3/2} \leq C^{\epsilon}$, then $\alpha(C) = C^{-\epsilon/2}$. This makes $S(C) \leq 2C^{\epsilon} + 20\ln(C)S(C^{1-\epsilon/2})$.

If we guess that $S(C) \leq 3C^{\epsilon}$, then we check the recursive equation and see that

$$S(C) \leq 2C^{\epsilon} + 20\ln(C) \cdot 3C^{\epsilon-\epsilon^2/2} \leq 3C^{\epsilon}$$

Observing that $C \leq n^2$ initially gives $S(C) \leq 3n^{2\epsilon}$, and we have proved the theorem. □

3 Clustering Algorithm

We return to Lemma . First, we will define some notation:

$B(i,r) = \{j \in V : d(i,j) \leq r\}$
$E(i,r) = E(B(i,r))$
$\delta(i,r) = \delta(B(i,r))$

where $d(i,j)$ is the length of the shortest i-j path in G.

Algorithm 2: Cluster

```
if α ≥ 1 then
    return each vertex in its own cluster
l ← 1
while G nonempty do
    Pick i in G
    Let r* be smallest r s.t. c(E(i, r+1)) ≤ (1+α)c(E(i,r))
    V_l ← B(i, r*)
    l ← l + 1
    Remove U_l and all incident edges from G
return V_1, ··· , V_k
```

Proof of Lemma 3:

We will prove both claims.

First, we must show that there are at most $\alpha(C)|E|$ intercluster edges. For each cluster U_l,

$$\begin{aligned} c(\delta(U_l)) &= c(\delta(i, r^*)) \\ &\leq c\Big(E(i, r^*+1) - E(i, r^*)\Big) \\ &= c(E(i, r^*+1)) - c(E(i, r^*)) \\ &\leq \alpha c(E(i, r^*)), \end{aligned}$$

by construction. So we can charge edges in $\delta(U_l)$ against edges in $E(i, r^*)$. The charge is unique since $E(i, r^*)$ is removed from the graph in the next step. Therefore, we have at most $\alpha(C)|E|$ intercluster edges.

Second, we claim that $r^* \leq \dfrac{D(C)}{2}$. Suppose not. Then,

$$c(E(i, r+1)) > (1+\alpha)c(E(i, r)) \qquad \text{for } r = 1, \cdots, \frac{D(C)}{2}$$

Therefore

$$\begin{aligned} c(E(i, r^*)) &\geq c(E(i, 1))(1+\alpha)^{D(C)/2} \\ &> 1\Big(e^{\alpha/2}\Big)^{D(C)/2} \\ &= e^{\frac{\alpha D(C)}{4}} && \text{using } 1 + x > e^{x/2} \text{ for } 0 < x < 1 \\ &= e^{\ln C} \\ &= C \end{aligned}$$

This is a contradiction, since there are more edges in $c(E(i, r^*))$ than there are in the entire graph. □

4 Final Remarks

For the overall running time, we notice that the clustering algorithm runs in $O(m)$ time, since it essentially does a breadth-first search. We observe that $\log(C)$ decreases by a factor of $1 - \frac{\epsilon}{2}$ at each recursive call, so that we make $O(\frac{1}{\epsilon}\log\log C) = O(\frac{1}{\epsilon}\log\log n)$ total recursive calls. This means the entire algorithm takes $O(\frac{1}{\epsilon}m\log\log n)$ time.

The best known result finds a tree of $O(m\log n\log\log n)$ total stretch in $O(m\log n\log\log n)$ time (Abraham and Neiman STOC 2012).

An open question is whether it is possible to find a tree of total stretch $O(m\log n)$ in $\tilde{O}(m)$ time. Alon et al. show that every tree has total stretch $\Omega(m\log n)$ in grid graphs and in graphs with $O(n)$ edges and girth $\Omega(\log n)$, so it is not possible to find a tree of total stretch $o(m\log n)$ in every graph.