Sixth Cargese Workshop on Combinatorial Optimization September 14, 2015

Lecture 1: Low-stretch trees

Nick Harvey *University of British Columbia*

Abstract

The main theme of the workshop is fast algorithms, particularly those that relate to fast solvers for linear systems involving the Laplacian of a graph. In my lectures, I will discuss three key technical ingredients that underlie those solvers.

In this first lecture, I will discuss "low-stretch trees". Given a graph, the goal is to find a spanning subtree such that, on average, distances in the tree approximate distances in the graph. These trees have many uses, but we will eventually see that they are particularly useful as preconditioners of Laplacian matrices.

1 Introduction

Many optimization problems on graphs are easier to solve when the graph is a tree. For example, the problem might have a unique or obvious solution on a tree, or the tree structure might facilitate a solution based on dynamic programming. This naturally leads to the idea of trying to approximate an arbitrary graph by a tree. This idea has proven to be useful in many contexts, including approximation algorithms, online algorithms, and solving systems of linear equations.

There are several variants of this problem and notions of approximation that are valuable to consider. The particular form of the problem discussed in this lecture is as follows:

- the input graph is undirected, simple, and unweighted,
- the goal is to minimize the distance in the tree, averaged over all adjacent pairs of vertices in the graph,
- the tree must be a spanning subtree of the graph (no additional edges or vertices),
- the algorithm must output a single tree, not a distribution.

Trees of this sort are called "low-stretch trees", and they are very useful as preconditioners for symmetric diagonally-dominant systems of linear equations, as was pointed out by Boman and Hendrickson in 2001. Over the past 15 years there has been tremendous progress on solving such linear systems, and almost all of those results rely on low-stretch trees.

We will present a classic result of Alon, Karp, Peleg and West that gave the first non-trivial construction of low-stretch trees. They also present a solution for the scenario in which the input graph's edges can have arbitrary lengths, but we will not have time to discuss those additional details.

2 Preliminaries

Let $G = (V, E)$ be an unweighted, undirected graph. Let $n = |V|$ and $m = |E|$. Let $d(u, v)$ denote the distance (number of edges on a shortest path) between vertices u and v. Let $B(v, i)$ be the vertices

within distance i of v. Let $E(v,i)$ be the edges with both endpoints in $B(v,i)$. Let $\delta(v,i)$ be the edges with exactly one endpoint in $B(v,i)$. The diameter of a graph is $\max_{u,v} d(u,v)$.

It will be useful to also consider unweighted multigraphs. Since parallel edges are indistinguishable, it will be convenient to simply record the number of parallel edges with the same endpoints. For that purpose, let c_e be a positive integer denoting the number of parallel copies of e. This will be called the "capacity" or "multiplicity" of e. For a subset $F \subseteq E$, let $c(F) = \sum_{e \in F} c_e$.

2.1 Low-diameter decompositions

One of the main techniques we will use is a ***low-diameter decomposition*** of a graph. This is a fundamental idea that has played a role in distributed computing, approximation algorithms, metric embeddings, etc.

A ***partition*** of a graph $G = (V, E)$ is a partition of V into parts $U_1 \cup U_2 \cup \cdots$ that are pairwise disjoint. A ***cluster*** is the subgraph of G induced by one of the U_i. An edge of G is called an ***internal*** edge if both endpoints belong to the same cluster, otherwise it is called an ***inter-cluster*** edge.

Lemma 1. Let $G = (V, E)$ be a graph with positive integer edge multiplicities c_e. Let $C = c(E)$ and let D be an arbitrary parameter. There is a partition of G such that

- every cluster has diameter at most D,
- the fraction of edges that are inter-cluster is $\alpha = 4\ln(C)/D$, taking multiplicities into account.

References: Awerbuch 1985 Section 4, Leighton-Rao 1999 Section 2.2, Shmoys-Williamson 2010 Lemma 8.7.

3 Low-stretch trees

Let $G = (V, E)$ be an undirected, simple, unweighted graph. Let T be a spanning subtree of G. The notation $d_T(u, v)$ is the number of edges in the unique path between u and v in T. The ***average stretch*** of T is defined to be

$$\frac{1}{|E|} \sum_{(u,v) \in E} d_T(u, v)$$

More generally, if G has edge multiplicities given by $c : E \to \mathbb{N}$, then the stretch is defined to be

$$\frac{1}{c(E)} \sum_{(u,v) \in E} c_e \cdot d_T(u, v).$$

(The distance $d_T(u, v)$ does not depend on c.)

Theorem 2. Let $\epsilon > 0$ be any fixed constant. Let G be a graph with n vertices and no edge multiplicities. Then G has a spanning subtree with average stretch $n^{O(\epsilon)}$.

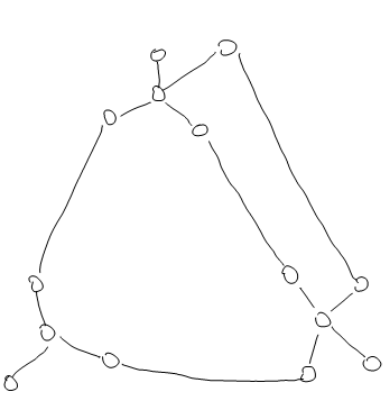
References: Alon-Karp-Peleg-West Section 5.2.

At a high level, their algorithm is a fairly natural recursive use of low-diameter decompositions. Pseudocode is shown in Algorithm 1. As we will see, one the the key details is to carefully choose the diameter D to be a function $D = D(C)$ of the total capacity $C = c(E)$.

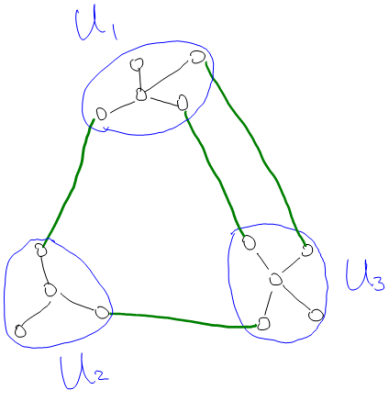
Let us now analyze the stretch of the resulting tree. Consider any edge $\{v, w\} \in E$. Let P be the path between v and w in T. In general P may contain edges from T' and edges from several T_i.

Algorithm 1: An algorithm for finding a low-stretch spanning subtree of G.

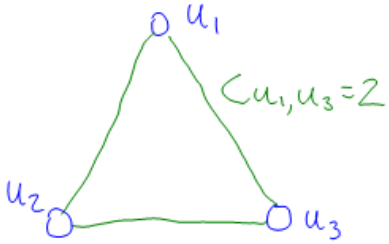
Function `LowStretchTree`$(G = (V, E), c)$**:**
- Let $C = c(E)$.
- Call `BuildPartition` to find a low-diameter decomposition of G (with multiplicities c) and diameter $D = D(C)$. The resulting clusters are $U_1, U_2, \ldots$
- Compute a shortest-path tree T_i within each cluster U_i, rooted arbitrarily.
- Construct graph G' by contracting each cluster U_i into a super-vertex u_i. Let the vector c' record the multiplicities of any parallel edges that are created.
- $T' \leftarrow$ `LowStretchTree`(G', c')
- Return $T \leftarrow T' \cup \bigcup_i T_i$. (An edge in T' with multiplicity greater than one is mapped to an arbitrary pre-image in G.)



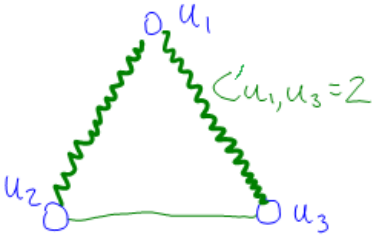
(a) G



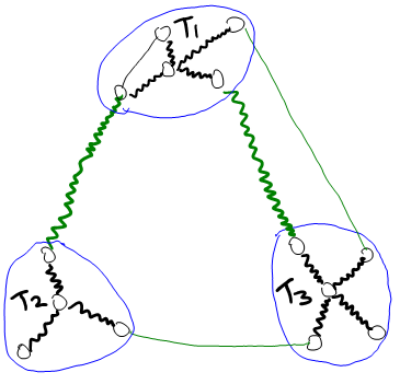
(b) Low-diameter decomposition of G. Green edges are inter-cluster.



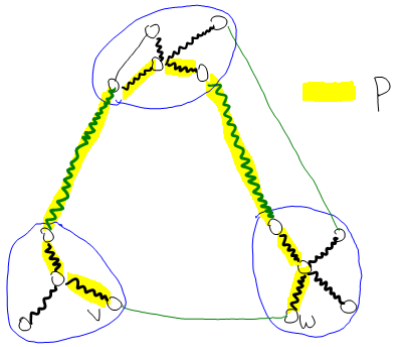
(c) Contracted graph G'



(d) Spanning tree T'



(e) Spanning tree T



(f) Path between v and w in T

Figure 1: Example.

Case 1: Internal edge. First suppose $\{v, w\}$ is internal to a cluster U_i. Since the diameter of each cluster is at most D and T_i is a shortest path tree within that cluster, then P has length at most $2D$.

Case 2: Inter-cluster edge. Otherwise $\{v, w\}$ is an inter-cluster edge, say $v \in U_a$ and $w \in U_b$. Then P is formed from a path P' in T' between the super-vertices u_a and u_b, together with paths in T_i for every super-vertex u_i traversed by P'. As before, the portions within each T_i have length at most $2D$. The main question is the length of P'.

The distribution on pairs $\{u_a, u_b\}$ induced by picking an *inter-cluster* edge in G proportional to the multiplicities c is the same as when picking an edge in G' proportional to the multiplicities c'. Under the latter distribution, the expected number of edges of the u_a-u_b path in T' equals the average stretch of T'. In this case the expected number of edges in P is at most

$$\begin{aligned}
&(\text{expected \# edges in } P') \;+\; (\text{expected \# super-vertices in } P') \cdot (\text{max diameter of any tree } T_i) \\
=\;& (\text{avg. stretch of } T') \;+\; \big((\text{avg. stretch of } T') + 1\big) \cdot 2D(C) \\
\leq\;& (\text{avg. stretch of } T') \cdot 5D(C)
\end{aligned}$$

Recurrence Analysis. This leads to a recurrence for the stretch of T. Let $s(C)$ be the maximum, over all graphs with total capacity C, of the average stretch returned by this algorithm. The properties of the low-diameter decomposition ensure that the fraction of inter-cluster edges is at most $\alpha(C) = 4\ln(C)/D(C)$. Consequently, the total capacity of G' is at most $\alpha(C) \cdot C$. This leads to the recurrence

$$\begin{aligned}
s(C) &\leq 2D(C) + \alpha(C) \cdot s(\alpha(C) \cdot C) \cdot 5D(C) \\
&\leq 2D(C) + 20\ln(C) \cdot s(\alpha(C) \cdot C) \qquad (1)
\end{aligned}$$

Now define $D(C) = 4\ln(C)C^{\epsilon/2} \leq C^{\epsilon}$, so that $\alpha(C) = C^{-\epsilon/2}$. We prove by induction that $s(C) \leq 3C^{\epsilon}$, for sufficiently large C. We have

$$\begin{aligned}
s(C) &\leq 2C^{\epsilon} + 20\ln(C) \cdot s(C^{1-\epsilon/2}) \\
&\leq 2C^{\epsilon} + 60\ln(C) \cdot C^{\epsilon - \epsilon^2/2} \\
&\leq 3C^{\epsilon}.
\end{aligned}$$

As all edges in the original graph G have multiplicity one, we have $C \leq n^2$. This completes the proof of Theorem 2.

4 Low-diameter decompositions

Let the total capacity be $C = c(E) = \sum_{e \in E} c_e$. Our goals are to find a partition such that:

- every cluster has diameter at most D
- the total capacity of the inter-cluster edges is at most $\alpha = 4C \ln(C)/D$.

The main idea is to use breadth-first search from a single node to find a maximal cluster whose diameter is roughly the logarithm of the size of the cluster. Maximality ensures that the number of edges leaving the cluster is a small fraction of the internal capacity. Choosing parameters carefully ensures that the diameter is at most D.

We claim that the following algorithm produces the desired cluster.

Algorithm 2: Repeatedly build a cluster of diameter at most D such that at most an α-fraction of the edges are inter-cluster.

Function `BuildCluster`$(G = (V, E), c, \alpha)$**:**
  Let v be an arbitrary node in G.
  Let $i^* = \operatorname{argmin}\{\, i \geq 1 : c(\, E(v, i+1)\,) < (1+\alpha) \cdot c(\, E(v,i)\,) \,\}$.
  Return $B(v, i^*)$
Function `BuildPartition`$(G = (V, E), c, D)$**:**
  Let $C = c(E)$ and $\alpha = 4 \ln(C)/D$.
  if $\alpha \geq 1$ **then**
    Return the trivial partition (V)
  repeat
    Let $U \leftarrow$ `BuildCluster`(G, c, α) be a new cluster.
    Delete the vertices in U from G. (Don't update C or α.)
  until G is empty;

Analysis of inter-cluster edges. Consider a single execution of `BuildCluster`. The new inter-cluster edges created in that execution have capacity

$$c(\, \delta(v, i^*)\,) \;\leq\; c\big(\, E(v, i^*+1) \setminus E(v, i^*)\,\big) \;=\; c\big(\, E(v, i^*+1)\,\big) - c\big(\, E(v, i^*)\,\big) \;\leq\; \alpha \cdot c(\, E(v, i^*)\,),$$

by choice of i^*. Thus, summing over all clusters, the total capacity of all inter-cluster edges is at most an α fraction of the total capacity of all edges, since clusters are vertex-disjoint.

Diameter of clusters. We claim that $i^* \leq D/2$. If not, we would have $c(\, E(v, i+1)\,) \geq (1+\alpha) \cdot c(\, E(v,i)\,)$ for $i = 1, \ldots, D/2$. Since capacities are positive integers, $c(\, E(v,1)\,) \geq 1$. This implies that

$$c(\, E(v, i^*)\,) \;\geq\; c(\, E(v,1)\,) \cdot (1+\alpha)^{D/2} \;>\; \exp(\alpha/2)^{2\ln(C)/\alpha} \;=\; C,$$

which is a contradiction. Therefore each cluster has radius at most $D/2$ and diameter at most D.

5 Further discussion

5.1 Running time

As mentioned, low-stretch trees form a key ingredient in fast Laplacian solvers, so we ought to be able to find these trees quickly. The `BuildPartition` procedure is essentially performing breadth-first search,

so it can be implemented in $O(m)$ time. The work of the `LowStretchTree` procedure is dominated by calling `BuildPartition` then computing the T_i trees (again by breadth-first search), so this can be implemented in $O(m)$ time. The number of levels of recursion is $O(\frac{1}{\epsilon}\log\log C)$ because $\log C$ decreases by a factor $1-\epsilon/2$ in each level. So the overall runtime is $O(\frac{1}{\epsilon}m\log\log n)$.

5.2 An improvement: sub-polynomial stretch

The low-stretch tree analysis can be improved to get stretch that is sub-polynomial (but not quite polylog). Pick $D(C) = \exp\big((\ln C)^{0.51}\big)$. We will prove by induction that $s(C) \le 3D(C)$. By (1), it suffices to show that

$$20\ln(C)\cdot s\Big(\frac{4C\ln C}{\exp\big((\ln C)^{0.51}\big)}\Big) \;\le\; D(C).$$

By induction, the left-hand side is at most

$$\begin{aligned} &60\ln C\cdot\exp\left(\Big(\ln\frac{4C\ln C}{\exp\big((\ln C)^{0.51}\big)}\Big)^{0.51}\right) \;\le\; 60\ln C\cdot\exp\left(\Big(\ln C-(\ln C)^{0.51}/2\Big)^{0.51}\right)\\ \overset{(a)}{\le}\;& 60\exp(\ln\ln C)\cdot\exp\left((\ln C)^{0.51}-(\ln C)^{0.02}/4\right) \;\le\; D(C), \end{aligned}$$

for sufficiently large C. The definition of $D(C)$ is primarily dictated by step (a), which uses that $(x-x^b)^b \approx x^b - bx^{2b-1}$. We need $2b-1>0$ (in order to cancel off the other unwanted factors), leading to the choice $b=0.51$.

5.3 Further improvements

Spanning subtrees with stretch $O(\log^2 n\log\log n)$, and later with stretch $O(\log n\log\log n)$, have been developed[1]. These results are also based on recursively decomposing the graph, but instead of using low-diameter decompositions they respectively use the "star decomposition" and the "petal decomposition". It is conjectured that spanning subtrees with stretch $O(\log n)$ exist. Alon et al. showed that every tree has stretch $\Omega(\log n)$ in (a) any graph with $O(n)$ edges and girth $\Omega(\log n)$, and (b) a square grid.

References: Elkin et al. 2008, Abraham-Bartal-Neiman 2008, Abraham-Neiman 2012, Papp 2014.

References

[1] I. Abraham, Y. Bartal, and O. Neiman. Nearly tight low stretch spanning trees. In *FOCS*, 2008.

[2] I. Abraham and O. Neiman. Using petal-decompositions to build a low stretch spanning tree. In *STOC*, 2012.

[3] N. Alon, R. M. Karp, D. Peleg, and D. West. A graph-theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 24(1):78–100, 1995.

[4] B. Awerbuch. Complexity of network synchronization. *Journal of the ACM*, 32(4):804–823, 1985.

[5] M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng. Lower-stretch spanning trees. *SIAM Journal on Computing*, 38(2):608–628, 2008.

[6] T. Leighton and S. Rao. Multicommodity Max-Flow Min-Cut Theorems and Their Use in Designing Approximation Algorithms. *Journal of the ACM*, 46(6):787–832, Nov. 1999.

[7] P. A. Papp. Low-stretch spanning trees, 2014. BSc thesis, Eötvös Loránd University.

[8] D. B. Shmoys and D. P. Williamson. *Design of Approximation Algorithms*. Cambridge University Press, 2010.

[1] Papp observes some flaws in the claims of Elkin et al.