CS 594: Representations in Algorithm Design **Spring 2022**

Lecture 6: 01/27/2022

Lecturer: Xiaorui Sun *Scribe: Ian P. Swift*

1 Last time and today

Last lecture, we were introduced to the concept that a tree T can represent a graph $G(V, E)$, such that certain desirable properties about the graph are reflected within the representation.

In particular, we introduced the concept of "Stretch". Given and edge from the original graph $e_{u,v} \in E$, the stretch of an unweighted edge is the distance between u and v in T: $Stretch_T(e) = dist_T(u, v)$. We express the stretch of T across the entire graph is the sum of the stretch of all the edges $Stretch_T(G) = \sum_{e \in E} Stretch_T(e)$.

The goal is to find a tree s.t., $Stretch_T(G)$ is small. This is called a "low stretch tree", or in the case that the generated tree is a spanning tree, a "low stretch spanning tree". In later sessions, we will discuss how a randomly generated tree can serve as a good approximation for the distance between *any* two nodes.

Today, we will show a method known as Low Diameter Decomposition (LDD) which can be used to generate trees which efficiently meet our goal. Once we have the LDD, we will use it to generate a Low Stretch Spanning Tree.

2 Low Diameter Decomposition (LDD)

In order to achieve a low stretch spanning tree, we first attempt to partition the graph into small pieces s.t., the diameter within the pieces is small. The intuition is that by reducing the diameter in these subcomponents, the overall stretch will be smaller. We call such a partition a Low Diameter Decomposition (LDD).

A partitioning of a graph into sets $S_1, S_2, ...S_k$ is considered a LDD if it holds the following properties:

- Within each set S_i the diameter of $G(S_i) \leq \Delta$
- The number of crossing edges is $<<$ the total number of edges in the graph.

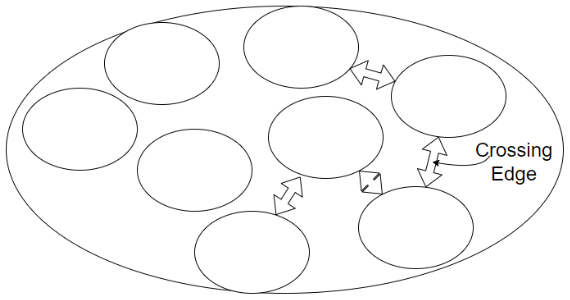

Figure 1: A graph to be decomposed containing several subgraphs, connected by "connecting edges"

By performing a LDD, our goal is to reduce the problem of finding a Low Stretch Spanning Tree into smaller problems which are easier to solve than finding the tree for the overall graph. Since this is a graph problem, the subcomponents (subgraphs) will still be connected by "bad" edges. However, by limiting the number of "bad" edges, we create problems that are easier to solve here as well.

As a side note, this general idea of "graph decomposition" will work for other problems as well. The only requirement is that the property wanted to be upheld is upheld by the subgraphs the graph is decomposed into. The intent then is that the solution for the decomposed "special case" is a solution for the "general case" that the implementer is initially trying to solve.

2.1 Formal Definition

Lemma 1 *For an unweighted graph G, there exists a polynomial time algorithm to find a LDD $S_1, ..., S_k$ s.t.,*

- *$G(S_i)$ has a diameter $\leq \Delta$*
- *The number of crossing edges is bound by $O(\frac{m \times log(n)^2}{\Delta})$*

In particular, by defining Δ to be $\sqrt{n}$ we find that we have a stretch of at most $O(m \times \sqrt{(n)}log(n)^2)$. This also means that by running the LDD with $\Delta = \sqrt{n}$ that we find the following properties:

- Diameter of $G(S_i) \leq \sqrt{n}$
- The number of crossing edges is bound by $O(\frac{m \times log(n)^2}{\sqrt{n}})$

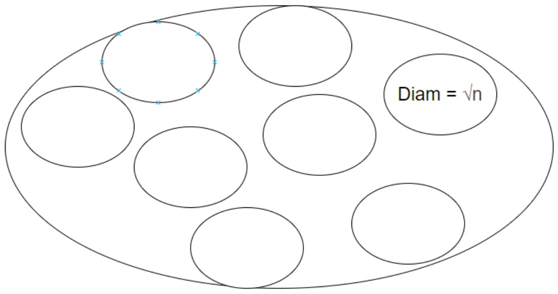

Figure 2: A LDD with Δ specified as $\sqrt{n}$ yielding components with diameter at most $\sqrt{n}$

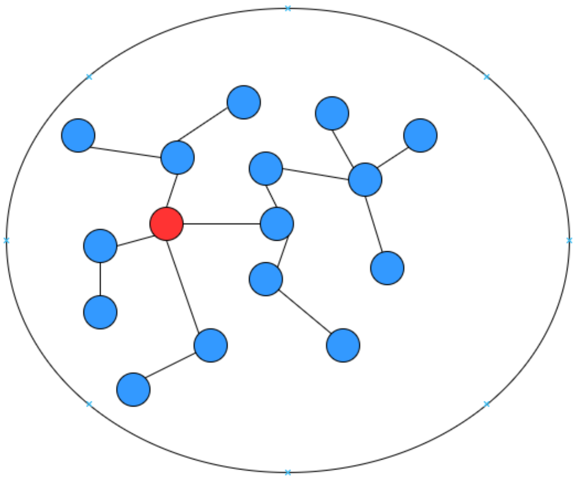

Figure 3: A subcomponent represented using a BFS tree

2.2 An LDD of $\Delta = \sqrt{n}$

In order to construct a Low Stretch Spanning Tree, first we will choose an arbitrary vertex and perform a Bredath First Search, generating a tree rooted at the node. We can guarantee a stretch of $\sqrt{n}$ by limiting the depth of the tree to $\sqrt{n}$. The distance from the root to any node is $\sqrt{n}$, so the distance between any two nodes in this tree is at most $2 \times \sqrt{n}$. It follows that the total stretch of two nodes within the same these trees is $O(m \times 2\sqrt{n}) \leq O(m \times 2\sqrt{n} \times log(n)^2)$

Next, to connect the trees, use arbitrary edges between $S_i, ..., S_k$ s.t., the connected. components form a tree. The distance between any two nodes on a tree that connects all nodes is at most n. Since the number of crossing edges is at most $\frac{m \times log(n)^2}{\Delta}$, and the stretch of each of these edges is n, it follows that the stretch for all these edges is $\leq \frac{m \times log(n)^2}{\sqrt{n}} \times n = m \times \sqrt{n} \times log(n)^2$.

These two values are the same yielding a stretch that is $O(m\times\sqrt{n}\times log(n)^2)$. These values converge because the chosen delta was $n^{0.5}$. Hit it been a larger or smaller power of n, they would not have converged.

2.3 A smaller stretch tree using LDD

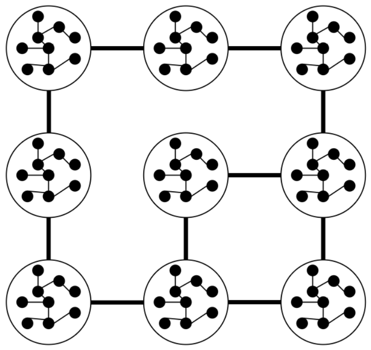
Figure 4: Nine subcomponents, each created using a BFS tree.

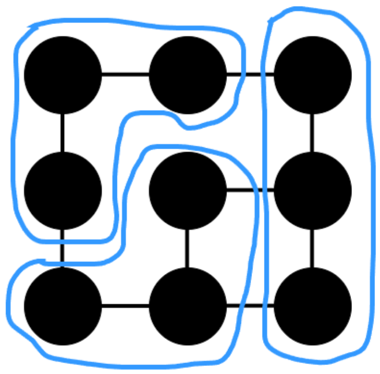
Figure 5: Three groups of three subcomponents, by BFS using edges connecting subcomponents.

Next we take an iterative approach to the problem in order to achieve a smaller stretch. In this version, we will set $\Delta = n^{1/3}$. After performing the first LDD, we achieve the series of subcomponents and connecting edges depicted in Figure 4. For this LDD, we have the properties: diameter of $G(S_i) \leq n^{1/3}$ and the number of crossing edges is bound by $O(\frac{m\times log(n)^2}{n^{1/3}})$.

Next we take each subcomponent and reduce it to a single vertex R_i. Using this reduced graph we again perform a LDD with $\Delta = n^{1/3}$. In these contracted subcomponents we see the following property. The distance between two points u, v in a contracted subgraph is equal to the number of edges in the contracated graph ($\leq n^{1/3}$) times the size of the connected subgraphs ($\leq n^{1/3}$). Meaning that the distance between any vertices edges in the contracted subgraphs is $\leq n^{2/3}$.

We now look at the contribution of various edges. Since the distance of a crossing edge in the first LDD is at most $n^{2/3}$ and the number of crossing edges is bound by $O(\frac{m\times log(n)^2}{n^{1/3}})$, the total contribution is bound by $O(m \times n^{1/3}log(n)^2)$. The contribution from crossing edges in the second LDD is n, since they can cover the entire graph. Additionally, the number of crossing edges in the second LDD is bound by $O(\frac{m\times log(n)^4}{n^{2/3}})$

meaning that the contribution of these edges o the stretch is $O(m \times n^{1/3} log(n)^4)$. Finally, the original subcomponents generated had a stretch bound by $O(n^{1/3})$ the contributions of these is simply bound by $O(m \times n^{1/3})$.

Thus, since all the stretch contributions are within a polylog difference of each other, by repeating the process twice with a Δ of $n^{1/3}$ we see a stretch of $O(m \times n^{1/3} polylog(n))$. Equally you can repeat this process many times times. In doing so we see the following:

Lemma 2 *By setting $\Delta = n^{1/t}$ and repeating the LDD on a graph $t-1$ times, contracting the graph in between each LDD, a stretch is achieved bounded by $O(m \times n^{1/t} log(n)^{O(t)})$.*

By setting t $= \sqrt{\frac{log(n)}{loglog(n)}}$, the stretch is bounded by $O(m \times n^{\sqrt{\frac{loglog(n)}{log(n)}}} logn) = O(m \times 2^{\sqrt{log(n) \times loglog(n)}})$. It is demonstrable that this value is lower bounded by an arbitrarily large power of log(n) time and upper bounded by an arbitrarily small power of n time, meaning that this equation is $O(m * n^{o(1)})$

3 Next Time...

Another property that can be asked is how about for all vertex pairs. We will start with the assumption that for a graph $G(V, E, \omega)$ and for any two connected nodes u, v that $\omega(u, v) \leq \omega(u, x) + \omega(x, v) \forall x$.

For a distribution of tree $\mathcal{T}$, G is considered to be α embedded to $\mathcal{T}$ if:

- For each $T \in \mathcal{T}, d_T(u, v) \leq d_G(u, v)$
- $\mathbb{E}[d_T(u, v)] \leq \alpha \times d_G(u, v)$

Based on this, it will be demonstrated in the next class that $\exists$ randomized algorithm which ouputs trees s.t., G is $O(log(\Delta) \times log(n))$ embedded to $\mathcal{T}$.