1 Introduction

1.1 Motivation

Finding a low-stretch spanning tree of a graph is a graph theory problem that has many applications in communication networks [1] and in solving systems of linear equations [2]. Improvements to the stretch of the tree lead to faster algorithms for solving symmetric diagonally dominant linear systems, as pointed out first in [3]. Other applications of the low-stretch spanning tree are listed in section 1.1 of [2] and 1.4 of [4].

In the previous semester, the rapidly-exploring random tree (RRT) algorithm was introduced to find a low-stretch spanning tree. However, we found an example of a family of graph with a subset of edges that have high expected stretch. Adding multi-edges to this family of graph will increase the average stretch of the graph to $O(n^c)$ with constant $c < 1$. This project aims to improve the RRT algorithm by changing the sampling step to build low-stretch trees.

1.2 Definition

Consider a connected, undirected, weighted graph $G = (V, E, W)$, where $V$ is the set of vertices, $E$ the set of edges and $W$ the set of weights, we construct a spanning tree $T = (V, E_T)$ where $E_T \subseteq E$ and $|E_T| = |V| - 1$. For $(a, b) \in E$, let $d(a, b)$ be the weight of the edge $(a, b)$\footnote{For unweighted graph, the weight of all edges is 1.} Let the length of the shortest path connecting $a$ and $b$ in
be \( l(a, b) \). We define the stretch of each edge \((a,b)\) and of tree \(T\) as:

\[
\text{stretch}_{(a,b)} = \frac{l(a,b)}{d(a,b)} \quad (1)
\]

\[
\text{stretch}_T = \sum_{(a,b) \in E} \text{stretch}_{(a,b)} = \sum_{(a,b) \in E} \frac{l(a,b)}{d(a,b)} \quad (2)
\]

In this project, we want to find the tree \(T\) such that \(\text{stretch}_T\) is low.

### 1.3 Rapidly exploring random tree

**Data:** \(G = (V,E)\)

**Result:** \(G' = (S,P)\) with \(S = V\) and \(P \subseteq E\) and \(G'\) is a spanning tree.

Pick a random vertex to be the root and add it to the set \(S\);

\[\text{while } \|S\| \leq n \text{ do} \]

\[\text{Sample a vertex } v \in V \text{ uniformly randomly;}\]

\[\text{Let } P_{v,S} \text{ be a shortest path from } v \text{ to } S. \text{ Let } u \in P_{v,S} \text{ be the vertex on this}\]

\[\text{path that is adjacent to } S;\]

\[\text{Add } u \text{ to } S \text{ and the corresponding edge on } P_{v,S} \text{ to the tree we are}\]

\[\text{maintaining.}\]

\[\text{end}\]

**Algorithm 1:** RRT algorithm

### 1.4 Counter example

We build a graph \(G = (V,E,W)\) such that the expected stretch of an edge in a tree
constructed by RRT algorithm is \(\Omega(n^c)\). This example was provided by Anup Rao [5].

Consider a ring graph \(R_{l+1} = (v_0, v_1, ..., v_l)\), with the lengths of the edges to be:
\(d(v_i, v_{i+1}) = 2^i, 0 \leq i \leq l - 1\) and \(d(v_l, v_0) = 1\). We also add perturbation to break ties between \(d(v_i, v_{i+1})\) and \(d(v_i, v_l)\) such that \(d(v_i, v_{i+1}) = 2^i + \epsilon, 0 \leq i \leq l - 1\) and \(d(v_l, v_0) = 1 + \epsilon\). Note that in the tree \(T = R_{l+1} \setminus (v_0, v_l)\), the stretch of \((v_0, v_l)\) is \(2^l\).

For each \(v_i\), create \(k^i\) copies of \(v_i\), denoted as \(v_{i1}, v_{i2}, ..., v_{ik}\), where \(v_{ij}\) denotes the \(j^{th}\) copy of \(v_i\). Pick \(k\) and \(l\) such that \(1 + k + k^2 + ... + k^l = n\). For any two vertices which are copies of \(v_i\) or is \(v_i\), add an edge between them with distance \(\epsilon\). Then, for any copies of \(v_i\) and \(v_j\), add an edge of length \(d(v_i, v_j)\) between them. \(\forall i, j, k; d(v_{i1}, v_{j1}) = \epsilon\) and \(\forall i, j, k, l; d(v_{ik}, v_{jl}) = d(v_i, v_j)\). Add these vertices and edges to \(G\).

Running the RRT algorithm on this graph \(G\) gives a spanning tree in which \((v_0, v_i), \forall i \leq l\) has an expected stretch of \(\Omega(n^c), c \leq 1\).
1.5 Motivation: weighted ring graphs

Running the RRT algorithm on a weighted ring graph gives the probability of cutting an edge that is not directly related to its length. The following algorithm on a weighted ring graph, with a different step of adding vertex, gives us the desired relationship between the probability of cutting an edge and its length.

**Data:** \( G = (V, E, W) \) is a weighted ring graph.

**Result:** \( G' = (S, E', W') \) with \( S = V \) and \( E' \subseteq E, W' \subseteq W \) and \( G' \) is a spanning tree.

Pick a random vertex to be the root and add it to the set \( S \).

**while** \( \|S\| \leq n \)**

<table>
<thead>
<tr>
<th>Sample a vertex ( v \in V - S ) uniformly randomly.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let ( P_1 ) be the shortest path from ( v ) to ( S ) with total path length ( l_1 ). Let ( u_1 ) be the vertex on this path ( P_1 ) that is adjacent to ( S ).</td>
</tr>
<tr>
<td>Let ( P_2 ) be the second shortest path from ( v ) to ( S ) with total path length ( l_2 ). Let ( u_2 ) be the vertex on this path ( P_2 ) that is adjacent to ( S ).</td>
</tr>
<tr>
<td>Add ( u_1 ) to ( S ) and the corresponding edge on ( P_1 ) to the tree we are maintaining with probability ( \frac{l_1}{l_1 + l_2} ). Otherwise, ( u_2 ) is added.</td>
</tr>
</tbody>
</table>

**end**

**Algorithm 2:** Ring sampling algorithm

**Theorem 1.1** On a weighted ring graph \( G = (V, E, W) \), the probability of cutting each edge \( i \) using the ring sampling algorithm is \( d_i / n \), where \( d_i \) is the length of edge \( e_i \) and \( \sum_{i=1}^{n} d_i = n \).

Since the modification on the sampling scheme gives a nice structure on the weighted ring graph, we hypothesize that randomizing the choice of vertex to add in a similar manner will also help with the performance of the algorithm on the general graph.

2 My project

2.1 Goals

This project aims to:

1. Formalize an improved version of RRT algorithm.
2. Prove that the improved algorithm works on the counter example above.
3. Prove an upper bound for the stretch of the spanning tree produced by the improved algorithm and compare the performance of this algorithm to other known algorithms for computing low stretch spanning trees.
4. Implement a fast version of this algorithm in Julia.

2.2 Deliverables

1. A report containing the design and analysis of the improved RRT algorithm.
2. Source code in Julia.

References


[5] Anup Rao’s website is at [https://sites.google.com/site/anupraob/](https://sites.google.com/site/anupraob/)