Abstract

The performance of the Rapidly-exploring Random Tree (RRT) algorithm to find low-stretch spanning trees is analyzed on a special family of graphs. The construction of this graph family is described in the report, and we show that it is possible to force a group of edges to be removed with high probability. On this graph family, running the RRT algorithm gives a spanning tree with one group of edges that has expected stretch $O(n^c)$, $c < 1$. Adding multi-edges to the graph will lead to a counter example for the RRT algorithm because there exists at least one graph family such that running the RRT algorithm on this graph family gives a spanning tree of average stretch $O(n^c)$. Several ways to modify the algorithm are proposed. The idea of using random walk in the algorithm was proposed in Fall 2017 and a generalization of a random-walk step is investigated in this project. However, experimental evidence shows that a generalized random walk version of Rapidly-exploring Random Tree algorithm does not give trees with average stretch lower than $O(\log n)$. Another idea for changing the RRT algorithm by sampling vertices based on weighted degree also does not give trees with low average stretch. We also present another modified version of the RRT algorithm that may lead to a promising approach to find low-stretch trees. The new algorithm relies on the interpretation of graphs as systems of resistors or as flow networks, and the Laplacian of the graph is used to determine the sampling distribution on vertices.
1 Introduction

1.1 Low-stretch spanning tree

Let $G = (V, E, W)$ be a weighted, connected graph where $V$ is the set of vertices, $E$ is the set of edges, and $W$ is the set of weights. We construct a spanning tree $T = (V, E_T, W_T)$ where $E_T \subseteq E$ and $|E_T| = |V| - 1$. Following the notation in [3], we define the distance, or length of each edge $e = (a, b) \in E$ to be the reciprocal of its weight: $d(a, b) = d(e) = 1/w(e)$.

Let $d_G(a, b)$ be the sum of the edges on the shortest path from $a$ to $b$ in graph $G$. Since $T$ is a spanning tree, there exists a unique path $d_T(a, b)$ connecting $a$ and $b$.

We define the stretch of each edge $(a, b)$ and the stretch of tree $T$ as

$$\text{stretch}_{(a, b)} = \frac{d_T(a, b)}{d(a, b)},$$  \hspace{1cm} (1)

$$\text{stretch}_T = \sum_{(a, b) \in E} \text{stretch}_{(a, b)} = \sum_{(a, b) \in E} \frac{d_T(a, b)}{d(a, b)}. \hspace{1cm} (2)$$

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

1.2 Prior work

Many applications of spanning trees such as communication networks and distributed systems require the minimization of stretch instead of the sum of edge weights. The edge weights usually represent the importance of the connections or transmission delay in a system [1]. Finding the minimum average stretch tree has been shown to be NP-complete [4]. In this project, we only aim to decrease the upper bound for average stretch instead of finding the minimum stretch tree.

The paper by Alon, Karp, Peleg and West (1995) [2] led to the study of low-distortion graph embedding into probabilistic tree metrics. This first paper presented an algorithm that produces a spanning tree of stretch $2^{O(\sqrt{\log n \log \log n})}$. Ten years later, Elkin, Emek, Spielman and Teng [3] proposed the star-decomposition algorithm that produces significantly lower-stretch tree of $O((\log n \log \log n)^2)$ distortion. The idea of decomposing the graph into smaller components with nice properties such as low diameter or low radius is also seen in subsequent papers, such as in [6] (petal-decomposition) which gives a stretch of $O(\log \log n)$. Improvements to the distortion of the tree lead

---

1 For unweighted graph, the weight of all edges is 1.

2 Another variant of this problem in which we want to minimize the maximum stretch over all edges is also not easier, as it is shown in [5] that determining whether there exists a spanning tree $T$ with maximum stretch $\leq t$ for any real number $t > 4$ in weighted graph is NP-complete.
to faster algorithms for solving symmetric diagonally dominant linear systems, as pointed out first in [7]. Other applications of the low-stretch spanning tree are listed in section 1.1 of [3] and 1.4 of [8].

If the tree $T$ is not required to be a subgraph of $G$, for example, by allowing the addition of Steiner points or of edges that are not originally in $G$, then the average distortion of $O(\log n \log \log n)$ can be achieved, as shown in [8]. The algorithm in [9] improves on the result of [8] by a factor of $O(\log n \log \log n)$ to give an average stretch of $O(\log n)$. This result is tight, since there exist families of graphs that give $\Omega(\log n)$ expected stretch such as the grid graphs and graphs with $\Omega(\log n)$ girth [2][10].

2 Motivation for modifying the algorithm

First, we present the Rapidly-exploring Random Tree algorithm that was explored last semester.

2.1 Rapidly-exploring Random Tree algorithm

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

**Result:** $T = (S, E_T)$ with $S = V$ and $E_T \subseteq E$ and $T$ is a spanning tree.

Pick a random vertex to be the root and add it to the set $S$

while $|S| \leq n$ do

Sample a vertex $v \in V$ uniformly randomly

Let $P_{v,S}$ be a shortest path from $v$ to $S$. Let $u \in P_{v,S}$ be the vertex on this path that is adjacent to $S$.

Add $u$ to $S$ and the corresponding edge on $P_{v,S}$ to the tree we are maintaining.

end

**Algorithm 1:** RRT algorithm

2.2 Description of the graph family

We build a graph $G = (V, E, W)$ such that the expected stretch of one particular type of edge in a tree constructed by RRT algorithm is $\Omega(n^\epsilon)$. This example was provided by Anup Rao [14].

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 $d(v_l, v_0) = 1 + \epsilon$. The $\epsilon$ serves to break ties between $(v_l, v_0)$ and $(v_0, v_1)$. For subsequent description of the distance $d(v_l, v_0)$, we simply use 1 instead of $1 + \epsilon$. 
For example, we have:

\[
\begin{align*}
  d(v_l, v_0) &= 1 + \epsilon \\
  d(v_0, v_1) &= 2^0 = 1 \\
  d(v_1, v_2) &= 2^1 = 2 \\
  d(v_2, v_3) &= 2^2 = 4 \\
  \vdots \\
  d(v_{l-1}, v_l) &= 2^{l-1}
\end{align*}
\]

Note that in the tree \( T = R_{l+1} \setminus (v_0, v_l) \), the stretch of \((v_0, v_l)\) is almost \( 2^l - 1 \). We create duplicates of vertices to increase the probability of this edge being cut. We will define the graph \( G \) on \( n \) vertices by duplicating the vertices in \( R_{l+1} \) different number of times. For each \( v_i \), create \( k^i \) copies of \( v_i \), denoted as \( v_{ij}, v_{i2}, \ldots, v_{ii} \), where \( v_{ij} \) denotes the \( j^{th} \) copy of \( v_i \). Pick \( k \) and \( l \) such that \( 1 + k + k^2 + \ldots + k^l = n \). 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. \( \forall i, j, k, d(v_{ij}, v_{ik}) = \epsilon \) and \( \forall i, j, k, l, d(v_{ik}, v_{jl}) = d(v_i, v_j) \). Add these vertices and edges to \( G \).

An example of this graph family for \( k = 5, l = 4 \) is presented in Figure 1, where cluster \( i \) is represented by the number \( i \).
2.3 Analysis of the RRT algorithm on the graph family

We begin by showing that the RRT algorithm forces a group of edges to have high stretch. To prove this theorem, we use the following lemma that was introduced in CS490 Report of the Fall semester.

Lemma 2.1. Running the RRT algorithm on a complete graph produces a tree with an expected depth of $O(\log n)$.

Theorem 2.1. Running the RRT algorithm on the graph $G$ constructed in Section 2.2 gives a spanning tree in which the stretch of the edge of type $(v_0, v_l)$ is $\Omega(n^c)$, $c \leq 1$.

Proof. Let cluster $i$ refers to the group of vertices that are the $k^i$ copies of $v_i$, including $v_i$. When there exists a vertex $v_{i1}$ in cluster $i$ that is already added to the tree, if another vertex $v_{ik}$ in the same cluster is sampled, the shortest path from $v_{ik}$ to the tree is the path with one edge $(v_{i1}, v_{ik})$. The expected stretch of any edge connecting two vertices in the cluster is just the expected stretch obtained by running RRT algorithm on an unweighted complete graph. By the previous lemma, since the depth of the resulting tree has expected depth $O(\log n)$, the expected stretch of the tree is also $O(\log n)$. Thus, we only look at the first instance of picking a vertex from each cluster. Let this sequence be $(v_{\sigma_1}, v_{\sigma_2}, ..., v_{\sigma_l})$.

We will show that the probability that sequence of first appearance of each cluster
\((v_{\sigma_1}, v_{\sigma_2}, ..., v_{\sigma_l})\) coincides with the sequence \((v_l, v_{l-1}, ..., v_1)\) is at least \((1 - 1/k)^l\). The probability that a vertex in the cluster \(v_l\) is chosen as the first vertex in the spanning tree is:

\[
\frac{k^l}{1 + k + k^2 + ... + k^l} = \frac{k^l(k - 1)}{k^{l+1} - 1} = \frac{1 - k^{-1}}{1 - k^{-l-1}} \geq 1 - \frac{1}{k}
\]

Given that a vertex in the cluster \(l\) is in the tree, the probability that a vertex in the cluster \(l - 1\) is chosen next and not a vertex in the cluster \(i, 1 \leq i \leq l - 1\) is:

\[
\frac{k^{l-1}}{1 + k + k^2 + ... + k^{l-1}} = \frac{k^{l}(k - 2)}{k^{l} - 1} = \frac{1 - k^{-1}}{1 - k^{-l-2}} \geq 1 - \frac{1}{k}
\]

Similarly, the probability that vertex in the cluster \(i\) is added to the spanning tree after a vertex in cluster \(i + 1\) was added to the tree is:

\[
\frac{k^i}{1 + k + k^2 + ... + k^i} = \frac{k^i(k - 1)}{k^{i+1} - 1} = \frac{1 - k^{-1}}{1 - k^{-i-1}} \geq 1 - \frac{1}{k}
\]

Thus the probability that the first vertex from each cluster added to the tree is in the sequence: \((v_{\sigma_1}, v_{\sigma_2}, ..., v_{\sigma_l}) = (v_l, v_{l-1}, ..., v_1)\) is at least \((1 - 1/k)^l\).

This is the only case that is interesting for analysis because this sequence of adding the vertices to the tree gives the maximum expected stretch for edge \((v_0, v_l)\). In this case, when the permutation is \((v_l, v_{l-1}, ..., v_1)\), RRT algorithm returns the spanning tree that is a path graph, with one vertex from each cluster \(v_l, v_{l-1}, ..., v_1\), and duplicated vertices attached to each \(v_i\). The edge with highest stretch is \((v_0, v_{li})\), for some \(v_{li}\) in cluster \(l\), with stretch:

\[
\text{stretch}_{(v_0, v_l)} = \frac{1 + 2 + 2^2 + ... + 2^{l-1}}{1} = 2^l - 1
\]

The expected stretch of this edge is thus:

\[
\mathbb{E}[\text{stretch}_{(v_0, v_l)}] \geq (2^l - 1)(1 - \frac{1}{k})^l \geq (2 - \frac{2}{k})^l - 1
\]

Since \(1 + k + k^2 + ... + k^l = \frac{k^{l+1} - 1}{k - 1} = n\), if we have \(k > 1\) constant, then \(l \geq \frac{\log n}{\log k}\). The value of \(k\) that maximizes \((2 - 2/k)^{\frac{1}{\log k}}\) is 4.4. When \(k = 4\), stretch of \((v_0, v_l)\) is at least \(1.34^{\log(n)} - 1 = n^{0.29} - 1\).

\[\square\]

**Corollary 2.1.1.** When the edge of type \((v_0, v_l)\) is not in the spanning tree but the stretch of other edges is low (less than \(c \cdot \log n\)), the average stretch of the tree is at most \(c \cdot \log n + 1\).
Proof. The edges \((a, b)\) in the graph family described in 2.2 can either have \((a, b)\) be in the same cluster or in adjacent clusters. If they are in the same cluster with size \(k^j\), the number of edges is \(k^j(k^j - 1)/2\). If they are in adjacent clusters \(k\) and \(k + 1\), the number of edges is \(k^j \cdot k^{j+1} = k^{2j+1}\). Thus the total number of edges in the graph is:

\[
|E| = \sum_{j=0}^{l-1} \frac{k^j(k^j - 1)}{2} + k^l \geq k^{2l} \tag{4}
\]

Since there are \(k^l\) edges of type \((v_0, v_l)\), and the stretch of each edge is very close to \(2^l - 1\), if we assume that the stretch of all other edges is small \((c \log n)\), then the average stretch of the tree is:

\[
\text{stretch} = \frac{k^l \cdot (2^l - 1) + (k^{2l} - k^l) \cdot c \log n}{k^{2l}} \tag{5}
\]

Since \(k \geq 2\), the equation simplifies to:

\[
\text{stretch} \leq \frac{k^l \cdot k^l - k^l + k^{2l} \cdot \log n - k^l \cdot c \log n}{k^{2l}} \leq c \log n + 1 \tag{6}
\]

\[\square\]

Corollary 2.1.2. If \(G'\) is constructed from \(G\) with multi-edges such that \((v_0, v_l)\) has multiplicity \(n\) for each \(v_l\) in cluster \(l\), and all other edges have multiplicity 1, then running RRT algorithm on \(G'\) gives an average stretch of \(O(n^{0.29})\).

Proof. Assume that all edges but \((v_0, v_l)\) have the smallest stretch possible. The total number of edges is bounded above by \(n^2 + k^l \cdot n\). Approximating \(k^l\) to be \(n \cdot (k - 1)/k\), the average stretch of the graph is:

\[
\text{stretch}_{G'} = \frac{n \cdot n^{0.29} \cdot k^l + n^2 - n}{n^2 + k^l \cdot n} \geq \frac{n^2 \cdot n^{0.29} + n^2}{n^2(1 + (k - 1)/k)} = O(n^{0.29})
\]

\[\square\]

We provide the result of running the RRT algorithm on a graph with \(k = 4, l = 5, n = 1365\) for 1000 trials and note the number of times the maximum stretch of an edge is 32 = \(2^5\), which occurs when an edge of weight 1 in the ring \(R_{l+1}\) is cut. The maximum stretch of the 1000 trials are sorted in order to easily observe the number of times the maximum stretch is close to 32.

The analysis predicts that when \(k = 4, l = 5, n = 1365\), the probability that the edge of type \((v_0, v_l)\) is not in the spanning tree is at least \((1 - 1/k)^l = 243/1024\). On 1000 trials in Figure 2, we observe that the maximum stretch of \(2^l = 32\) occurs in about 250 out of the 1000 trials. The experimental evidence matches our prediction.
3 Algorithms that did not work

3.1 Random walk

An algorithm that resembles taking a random walk on the ring graph was presented in the last section of CS490 Report of Fall 2017. An example of a random walk algorithm is Wilson’s loop-erased algorithm to generate a uniform spanning tree. [15] Since this algorithm returns a spanning tree chosen uniformly at random from the distribution of spanning trees, we cannot use this algorithm directly to find a low-stretch tree. We
attempt to generalize the algorithm on a general graph in the following.

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 $\|S\| \leq n$

| Sample a vertex $v \in V - S$ uniformly randomly. |
| For each neighbor $n_i$ of the tree $S$, let $P_i$ be the shortest path from $v$ to $S$ with total path length $l_i$ that passes through $n_i$. This means that $n_i$ is the vertex on this path $P_i$ that is adjacent to $S$. |
| Add $n_i$ to $S$ and the corresponding edge on $P_i$ to the tree we are maintaining with probability proportional to $1 - \frac{l_i}{\sum_j l_j}$. |

end

Algorithm 2: Random-walk sampling algorithm

If we allow for multi-edges, denote $m_i$ as the multiplicity of edge $i$. We change the probability of adding $n_i$ to $1 - m_i \cdot \frac{l_i}{\sum_j m_j \cdot l_j}$. The restriction of this algorithm on a weighted ring produces the same algorithm that was presented in section 4.1. of the Fall 2017 report. On a weighted ring, this algorithm produces a spanning tree in which the stretch of each edge is proportional to the distance of the edge.

However, experimental evidence shows that the stretch of the tree can be very large because the stretch of an edge with weight $\epsilon$ can be large. We did not pursue the analysis of this algorithm because the experimental evidence suggests that the algorithm does not produce low-stretch tree. The algorithm was run for 1000 trials on the graph constructed in [2.2] with $k = 4, l = 5$. The total number of vertices is 1365, and the multiplicity of every edge is one except for the $(v_0, v_l)$ edge with multiplicity 1000. The stretch of the graph obtained from the 1000 trials is sorted.

An intuitive explanation for the reason why the random-walk sampling algorithm in [2] does not give good stretch is that the probability of not including the $\epsilon$ edges in the spanning tree is high. This problem arises because we do not always add the edge on the shortest path (which is the $\epsilon$ edge) to the tree, and sometimes we choose the longer path with non-trivial probability. In the original RRT algorithm, we sample vertices with uniform probability and then pick the edge on the shortest path from the vertex chosen to the tree. In the random walk algorithm, we also sample vertices with uniform probability but sample the path back to the tree with a non-uniform distribution that favors the shorter paths. Since sampling the path to the tree gives high stretch tree, our next idea is to change the distribution in which we sample the vertices, and then pick the shortest path to the tree.
3.2 Sampling on weighted degree

The simplest modification for the probability distribution on vertices is to give higher probability to highly connected vertices. The weighted degree is thus used as a basis for the sampling step. The degree of a vertex is the number of edges incident to the vertex. The weighted degree is calculated as the sum of the weights (the reciprocal
of distance) of the edges incident to the vertex.

**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 $\|S\| \leq n$ do
  Sample a vertex $v \in V - S$ proportional to its weighted degree.
  Let $P_{v,S}$ be a shortest path from $v$ to $S$. Let $u \in P_{v,S}$ be the vertex on this path that is adjacent to $S$.
  Add $u$ to $S$ and the corresponding edge on $P_{v,S}$ to the tree we are maintaining.
end

**Algorithm 3:** Algorithm which samples vertices on weighted degree

We also test the performance of this algorithm on the graph family that forces the RRT algorithm to give high stretch to one edge. Similar to the previous section, the algorithm was run for 1000 trials on the graph constructed in 2.2 with $k = 4, l = 5$, and then the stretch values were sorted. The total number of vertices is 1365, and the multiplicity of every edge is one except for the $(v_0, v_l)$ edge with multiplicity 1000. The stretch reported below is the average stretch, since the maximum stretch is the stretch of the $\epsilon$ edge and is very large (greater than $10^{10}$).
4 A promising algorithm

Since the distribution on vertices based on the weighted degree does not give spanning trees with good stretch, we modify the distribution on vertices by incorporating information on the number of paths from one vertex to another in the following algorithm. Intuitively, the higher number of edges there is between two vertices, the more important it is to add that edge to the tree. One way to represent this information is to use the context of the electrical network to determine the probability of choosing vertices. We use the following example of a line graph to demonstrate how multi-edges change the voltage assignment. Denote the voltage at vertex $i$ as $V_i$. Fix the voltage at vertex 0 to be 0 and the voltage at vertex 2 to be 1 ($V_0 = 0, V_2 = 1$). We have $V_1 = 0.5$ and the current flow (given that the resistance of each edge is 1) is 0.5 in $\bar{5}$.

However, when we introduce 2 more multi-edges between vertex 1 and 2 and still fix $V_0 = 0, V_2 = 1$, the voltage at equilibrium of vertex 1 changes to 0.75, because the effective resistance between 1 and 2 is 1/3.

The voltage associated at each vertex is thus similar to the case where the distance
between 1 and 2 is only 1/3. This means that adding multi-edges are treated as increasing the importance of the edge (or decreasing the distance between two vertices). Ideally, when there are multi-edges like in Graph 2, we want to be able to treat those multi-edges like one single edge in Graph 3. Applying this intuition to the graph family described in 2.2 if there are \( n \) copies of the edge \((v_0, v_l)\), we want to treat the weight of this edge to be \( n \) times more, or equivalently, the distance \( d(v_0, v_l) = 1/n \), so that the probability of adding \( v_0 \) to the tree is higher.

When we treat a graph as a system of resistors, where resistor of edge \( e \) is \( r_e = d_e = 1/w_e \), solving for the induced voltages can be done by solving the equation \( \mathbf{v} = \mathbf{L}^{-1} \cdot \mathbf{i}_{\text{ext}} \). The solution of the system’s induced voltage when fixing the voltages of two vertices is also related to the problem of finding the flow at each vertex when there is one unit of flow (water, current, ...) from vertex \( v_a \) to \( v_b \). Mathematically, the induced flow of each vertex, \( \mathbf{f} \), when we flow one unit from vertex 1 to vertex 0 is:

\[
\mathbf{L} \cdot \mathbf{f} = (1, -1, 0, 0, ..., 0)^T \Rightarrow \mathbf{f} = \mathbf{L}^{-1} \cdot (-1, 1, 0, 0, ..., 0)^T
\]  

(7)

If we have only one vertex in the tree, we can flow one unit from each of the vertex not in the tree to the tree. Since \( \mathbf{L} \) is a linear operator, flowing one unit from each vertex outside the tree at the same time is equivalent to solving \( \mathbf{f} \) such that \( \mathbf{L} \cdot \mathbf{f} = (n - 1, -1, -1, -1, ..., -1)^T \). This vector \( \mathbf{f} \) is useful in sampling vertices, as we

\[\text{Figure 5: Graph 1: no multi-edge}\]

\[\text{Figure 6: Graph 2: multi-edges of (1, 2)}\]

\[\text{Figure 7: Graph 3: multi-edges treated as one single edge}\]

\[\text{Figure 8: Graph 4: induced voltage}\]

\[\text{Figure 9: Graph 5: induced flow}\]

\[\text{Figure 10: Graph 6: sampling vertices}\]
Figure 7: Graph 3: no multi-edges

see in the algorithm below:

**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 $S$.

Let $H$ be a copy of the graph $G$, with one vertex in $S$, and $n$ is the number of vertices in $H$. Let 0 be the vertex in $S$.

while $\|S\| \leq n$ do

1. Flow one unit from each vertex not in $S$ to $S$. For each vertex $v_i, v_i \notin S$, the score of each vertex is the sum of the amount of flow through that vertex. The final score $f_i$ of each $v_i$ is thus the sum of all the flows when we give one unit of flow from each vertex $v_i, v_i \notin S$ to $S$. Obtain $f$ by solving the system $L_H \cdot f = (n - 1, -1, ..., -1)^T$, where $n$ is the number of vertices in graph $H$.

2. Sample vertex $v$ with probability proportional to $1/(f_0 - f_i)$.

Let $P_{v,S}$ be a shortest path from $v$ to $S$. Let $u \in P_{v,S}$ be the vertex on this path that is adjacent to $S$.

Add $u$ to $S$ and the corresponding edge on $P_{v,S}$ to the tree we are maintaining.

Collapse $u$ with $S$ in the graph $H$ by setting all of the edges connecting $u$ to $S$ to have distance 0. Since we have collapsed all vertices in $S$ in the tree, let the only vertex in $S$ be 0.

end

**Algorithm 4:** Laplacian-based sampling algorithm

Step 1 in the sampling procedure is equivalent to solving the linear system of equation $Lx = b$, where $L$ in this context is the Laplacian matrix of the graph. Flowing one unit from $v_i$ to $v_j$ corresponds to $b$ being $(0,0,...,0,1,0,...,-1,0,0)$ where $b_i = -1$ and $b_j = 1$. Since we are adding the effect of each unit of flow from a different vertex and the system is linear, the combined value of flow is $-1$ at each vertex not in $S$, and $n - 1$ at the vertex $0 \in S$, where $n$ is the number of vertices in the graph $H$. The score of each vertex $f_i$ can be represented by the vector $f$ such that $f$ is the solution to the equation $A \cdot f = b$, where $b$ is $(n - 1, -1, -1, ..., -1)$ where $b_0 = n - 1$ and 0 is assumed to be the index of the vertex in $S$.

This Laplacian-based sampling algorithm gives good performance on the graph family in 2.2, as seen in Figure 8, with sorted stretch values.
On randomly generated weighted graphs, running this algorithm on graphs with 11000 vertices and random seed from 1 to 4000 also give an average stretch of $2 \log(11000)$ for each graph. A formal proof for why this Laplacian-based sampling gives low stretch for many graphs is not known, but this is a promising algorithm to be analyzed in the future.

References


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