A Simple Linear Time Algorithm for Computing a $(2k-1)$-Spanner of $O(n^{1+1/k})$ Size in Weighted Graphs

Surender Baswana * and Sandeep Sen **

Department of Computer Science and Engineering,
I.I.T. Delhi, Hauz Khas, New Delhi-110016, India.
E-mail:{sbaswana, ssen}@cse.iitd.ernet.in

Abstract. Let $G(V,E)$ be an undirected *weighted* graph with $|V| = n$, and $|E| = m$. A t-spanner of the graph $G(V,E)$ is a sub-graph $G(V,E_S)$ such that the distance between any pair of vertices in the spanner is at most t times the distance between the two in the given graph. A 1963 girth conjecture of Erdős implies that $\Omega(n^{1+1/k})$ edges are required in the worst case for any $(2k-1)$-spanner, which has been proved for $k = 1, 2, 3, 5$. There exist polynomial time algorithms that can construct spanners with the size that matches this conjectured lower bound, and the best known algorithm takes $O(mn^{1/k})$ expected running time. In this paper, we present an extremely simple linear time randomized algorithm that constructs $(2k-1)$-spanner of size matching the conjectured lower bound.
Our algorithm requires local information for computing a spanner, and thus can be adapted suitably to obtain efficient distributed and parallel algorithms.

Keywords: Graph algorithms, Randomized algorithms, Shortest path

1 Introduction

A spanner is a (sparse) sub-graph of a given graph that preserves approximate distance between all-pairs of vertices. In precise words, a sub-graph $G(V,E_S)$ is said to be a t-spanner of the graph $G(V,E)$ if, between any pair of vertices, the distance in the spanner is at-most t times the distance in the original graph. The value t is the *stretch factor* associated with the spanner.

The concept of a sparse spanner is motivated by numerous applications that involve computation of distances in a graph. Since the running time is proportional to the number of edges, therefore to achieve efficiency in computation time it is desired to have a sub-graph (of a given dense graph) that is sparse, but at the same time, preserves all-pairs distances approximately.

Spanners are used as underlying graph structures in various areas of distributed computing; e.g. the design of synchronizers [2] and design of succinct routing tables [9] implicitly generate spanners. Spanners are also used in computational biology [3] in the process of reconstructing phylogeny trees from matrices whose entries represent genetic distances among contemporary living species. For other numerous applications, please refer to the papers [1, 2, 9, 10].

1.1 Previous Work

Previously a number of papers [1, 6, 11] had addressed the problem of computing sparse spanners of graphs efficiently. In addition, a lot of work [9, 11] had also been done to establish a lower bound on the size of spanner in terms of stretch factor. To establish these bounds on the size of the

* Work was supported in part by a fellowship from Infosys Technologies, Bangalore.
** Work was supported in part by an IBM Faculty Partnership award

spanner, these results use the following relationship between the stretch of a spanner and girth of the graph.

A graph has girth at-least $t+2$ if and only if it does not have a t-spanner other than the graph itself.

A classical result from graph theory shows that every graph with $\Omega(n^{1+1/k})$ edges must have a cycle of size at most $2k$. It has been conjectured by Erdős [7], Bondy and Simonovits [5], and Bollobás [4] that this bound is indeed tight. Namely, for any $k \geq 1$, there are graphs with $\Omega(n^{1+1/k})$ edges that have girth greater than $2k$. However, the proof exists only for the cases $k = 1, 2, 3$ and 5. Since any graph contains a bipartite sub-graph with at least half the edges, the conjecture implies the existence of graphs with $\Omega(n^{1+1/k})$ edges and girth at-least $2k+2$. This bound and the relation between the stretch of a spanner and girth of given graph (mentioned above) imply a lower bound of $O(n^{1+1/k})$ on the size of $(2k-1)$-spanner.

For unweighted graphs, Halperin and Zwick [8, 12] gave an $O(m)$ time algorithm to construct a $(2k-1)$-spanner of size $O(n^{1+1/k})$. However, their algorithm does not seem to be extensible to weighted graphs. For weighted graphs, the first algorithm for constructing $(2k-1)$-spanners of $O(n^{1+1/k})$ size was given by Althöfer et al. [1]. However, the best known implementation of their algorithm has a running time of $O(mn^{1+1/k})$. Thorup and Zwick [11], improving a result of Cohen [6], gave a randomized algorithm for computing $(2k-1)$-spanner of optimal size in $O(kmn^{1/k})$ expected time. All the existing algorithms require computation of shortest distance information between many pairs of vertices [1], or computing shortest path trees from a set of $(n^{1/k})$ vertices [11]. Since there is a bound of $\tilde{O}(m)$ on the best known algorithm for shortest path trees, the running times of the earlier algorithms may not be able to achieve a bound of $O(m)$.

1.2 Our Contribution

We present a randomized algorithm that takes expected linear time for computing $(2k-1)$-spanner of optimal size. More specifically we show that

> Given a weighted graph $G(V, E)$, and integer $k > 1$, a spanner of stretch $(2k-1)$ and $O(kn^{1+1/k})$ size can be computed in expected time $O(km)$.

Unlike previous algorithms that require global information (computing full shortest path tree from few vertices or determining the pair-wise distance), our algorithm requires only local information, viz. in the neighborhood of each vertex or a group of vertices. In addition to achieving a linear time sequential static algorithm for computing spanner, the local approach of our algorithm leads to near optimal algorithms for computing a $(2k-1)$-spanner in distributed/parallel environment and in external memory as well. The algorithm can be suitably adapted to provide efficient partial dynamic algorithms for maintenance of spanners for small k as well.

We have organized the paper as follows. As a warm-up, we first present an $O(m)$ expected time algorithm for 3-spanner, and mention some of the key ideas (clustering of vertices) which we formalize and extend in order to compute a $(2k-1)$-spanner. We present an overview of the algorithm followed by the details and proofs of correctness of the algorithm. The details of the other applications are not given in this extended abstract.

2 Computing a 3-Spanner

In order to build 3-spanner of a graph (with potentially $\theta(n^2)$ edges), the objective is to minimize the number of edges (at most $O(n^{3/2})$) to be included in the spanner, and still ensure that the distance between any pair of vertices is not stretched beyond a factor of 3.

The algorithm selects edges to be included in the spanner in two phases. Without loss of generality, we can assume that the edge weights are distinct.

1. *Forming the clusters* :
   We form a sample $\mathcal{R} \subset V$ by picking each vertex independently with probability $n^{-\frac{1}{2}}$. The expected size of the sample set is $O(\sqrt{n})$. We group the set of vertices neighboring to these sampled vertices into clusters. Initially the clusters are $\{\{u\}|u \in \mathcal{R}\}$. Each $u \in \mathcal{R}$ will be referred to as the *center* of its cluster. We process a vertex $v \in V - \mathcal{R}$ as follows.
   - If v is not adjacent to any sampled vertex, we add all its edges to the spanner.
   - If v is adjacent to one or more sampled vertices, let $\mathcal{N}(v, \mathcal{R})$ be the sampled neighbor that is nearest to v. We add the edge $\mathbf{e}(v, \mathcal{N}(v, \mathcal{R}))$ to the spanner, and every other edge incident on v with weight less than the weight of the edge $e(v, \mathcal{N}(v, \mathcal{R}))$ to the spanner. The vertex v is added to the cluster centered at $\mathcal{N}(v, \mathcal{R})$.

   Finally, all the intra-cluster edges, i.e., the edges between the vertices belonging to the same cluster are removed.
2. *Joining the Clusters*:
   For each vertex v, we group all its neighbors into their respective clusters. There will be at-most $|\mathcal{R}|$ neighboring clusters of v. For each cluster adjacent to v, we add the least-weight edge among all the edge between (the vertices of) the cluster and v to the spanner.

It is easy to see that the above algorithm merely requires exploring the adjacency list of each vertex at-most twice (once in each of the two phases) in addition to picking a random sample of vertices. Thus the running time of the algorithm is $O(m)$. Let E_S^1 and E_S^2 be the sets of edges added to the spanner in the first phase and the second phase respectively.

From the description of the first phase of the algorithm, the following Lemma holds true.

Lemma 1. *For each edge $\mathbf{e}(u, v) \in E - E_S^1$, the edge from u to $\mathcal{N}(u, \mathcal{R})$ (the center of the cluster to which u belongs) has weight no more than the weight of the edge $\mathbf{e}(u, v)$.*

We shall us the following Lemma to show that the set $E_S^1 \cup E_S^2$ is a 3-spanner.

Lemma 2. *For an edge $e(u, v) \in E$ that is not present in the spanner $G(V, (E_S^1 \cup E_S^2)$ constructed above, the following assertion holds true:*
There is a path in the spanner with weight at-most three times the weight of the edge $e(u, v)$.

Proof. It follows from the first phase of the algorithm described above that both u and v must be adjacent to vertices of the sample $\mathcal{R}$. There are two cases now.
Case 1 : Both the vertices belong to same cluster, say the cluster centered at $w \in \mathcal{R}$. It follows from Lemma 1 that there is a 2-edge long path $u - w - v$ in the spanner with weight no more than twice the weight of the edge (u, v). (This provides a justification for deleting any intra-cluster edge from the set E at the end of first phase)
Case 2 : The vertices u and v belong to different clusters, and let u belongs to the cluster centered at $x \in \mathcal{R}$. Let $e(u', v)$ be the least weight edge $\in E - E_S^1$ among all the edges incident on v from the vertices of the cluster centered at x. It follows from the second phase of our algorithm, that the edge $e(u', v)$ was added to the spanner. Hence there is a path $u - x - u' - v$ from u to v in the spanner and its weight $\mathbf{w}_S$ can be bounded as follows.

$$\begin{aligned}
\mathbf{w}_S(u, v) &= \mathbf{w}(u, x) + \mathbf{w}(x, u') + \mathbf{w}(u', v) \\
&\leq \mathbf{w}(u, v) + \mathbf{w}(x, u') + \mathbf{w}(u', v) \quad \{\text{using Lemma 1 }\} \\
&\leq \mathbf{w}(u, v) + 2\mathbf{w}(u', v) \quad \{\text{using Lemma 1 }\} \\
&\leq 3\mathbf{w}(u, v) \quad \{\text{follows from the second phase of the algorithm}\}
\end{aligned}$$

Using the above Lemma, we can show that the spanner $G(V, E_S^1 \cup E_S^2)$ has stretch 3 as follows.

Consider any pair of vertices $u, v \in V$, and the shortest path p_{uv} between the two in the graph $G(V, E)$. For each edge e of this path that is missing in the spanner $G(V, E_S^1 \cup E_S^2)$, there is a path

in the spanner with weight at-most thrice the weight of the edge e (using Lemma 2). Applying this argument for each missing edge, we can conclude that there is a path between the vertex u and the vertex v in the spanner with weight at-most thrice the weight of the shortest path between the two in the graph $G(V, E)$.

Now we shall show that the size of the 3-spanner $E_S^1 \cup E_S^2$ as computed by our algorithm given above is bounded by $O(n^{3/2})$.

Note that the sample set $\mathcal{R}$ is formed by picking each vertex randomly independently with probability $\frac{1}{\sqrt{n}}$. It thus follows from elementary probability that for each each vertex $v \in V$, the expected number of incident edges with weight less than that of $e(v, \mathcal{N}(v, \mathcal{R})$ is $\sqrt{n}$. Thus the expected number of edges contributed to the spanner by each vertex in the first phase of the algorithm is $\sqrt{n}$. So the expected size of the set E_S^1 is $O(n^{3/2})$.

Remark Since we can verify the number of edges chosen in the first phase , we will repeat the sampling if the number of edges exceeds $O(n^{3/2})$ - the expected number of repetitions will be $O(1)$.

The number of the edges added to the spanner in the second phase (*Joining the Clusters*) is $O(n|\mathcal{R}|)$. Since the expected size of the sample $\mathcal{R}$ is $\sqrt{n}$, therefore, the expected size of E_S^2 is also $O(n^{3/2})$. Hence the expected size of the spanner $E_S^1 \cup E_S^2$ is $O(n^{3/2})$.

Thus we can conclude that 3-spanner of a weighted undirected graph can be computed in $O(m)$ expected time.

3 Key Ideas Underlying the Algorithm

The algorithm for computing a $(2k-1)$-spanner selects a set E_S of $O(kn^{1+1/k})$ edges from the given graph $G(V, E)$ ensuring that the following proposition holds true for each edge $e \notin E_S$.

> $\mathcal{P}_k(e)$: there is a path between the end-points of the edge e in the graph $G(V, E_S)$ consisting of at-most $(2k-1)$ edges, and the weight of each edge on this path is no more than that of the edge e

We say that $\mathcal{P}_k(v)$ holds for a vertex if $\mathcal{P}_k(e)$ holds true for each edge e incident on the vertex v. It follows from the discussion at the end of the previous section that the spanner formed in this way will be of stretch $(2k-1)$.

In order to pick a (small) set E_S of $O(n^{1+1/k})$ edges (from potentially $O(n^2)$ edges in a graph), that would ensure the proposition $\mathcal{P}_k(e)$ for each edge $e \in E - E_S$, the key idea underlying our algorithm is the partitioning of set of vertices into *clusters*. Recall from the previous section how the clustering of the vertices (by sampling a random set of vertices, and grouping each vertex with its nearest sampled neighbor) proves to be crucial in the computation of a 3-spanner with $O(n^{3/2})$ edges. It is the smaller number of these clusters (only $\sqrt{n}$) compared to the number of vertices (n) that enables us to get a bound on the size of the 3-spanner, and it is the closeness of the vertices within a cluster that ensured a bound on the stretch of the spanner. Note that the latter property (closeness of vertices of same cluster) is achieved by associating each vertex to its nearest sampled vertex in the clustering.

In order to design an algorithm for computing $(2k-1)$-spanner, we formally define the *clustering* of vertices, and associate a parameter called *radius of a cluster* that captures the closeness of the vertices of the same cluster compared to the vertices outside the cluster.

3.1 Definitions and Notations

The following definitions and notations are in context of a given weighted graph $G(V, E)$.

Definition 1. *A* **cluster** *is a subset of vertices. A partition of a set $V' \subset V$ into clusters is called* **clustering**.

Definition 2. *A set $S \subset E$ is a* **spanning set** *for a cluster $c \subset V$ if each pair of vertices of c is connected by a path that is internal to the cluster (i.e., the intermediate vertices of the path belong to the cluster c) and consists of edges from the set E_S only.*

Definition 3. *A cluster $c \subset V'$ with a spanning set $E_S \subset E'$ is a cluster of* **radius** $\leq i$ *in the graph $G(V', E')$, if there is a vertex $u \in c$ called the center of the cluster c such that the following holds true :*

> *For each each edge $\mathbf{e}(x, y) \in E' - E_S, x \in c$, there is a path from x to u internal to the cluster and consisting of at-most i edges each from the set E_S and having weight no more than the weight of the edge $\mathbf{e}(x, y)$.*

Intuitively the vertices of a cluster are close together compared to the vertices lying outside the cluster.

Definition 4. *A clustering $\mathcal{C}$ with a spanning set $E_S \subset E$ is a clustering of radius $\leq i$ if each of its cluster with spanning set E_S is a cluster of radius $\leq i$.*

We shall use the following notations in the rest of our paper.

- $\mathcal{E}(x, c_1)$: the edges from the set $\mathcal{E}$ that are incident from the vertices of cluster c_1 to the vertex x.
- $\mathcal{E}(c_1, c_2)$: the set of edges between vertices of cluster c_1 and vertices of cluster c_2 that belong to the set $\mathcal{E}$.
- $\mathcal{E}(\mathcal{S})$: the set of edges from set $\mathcal{E}$ between the vertices of the clusters of the set $\mathcal{S}$.
- $|\mathcal{C}|$: The number of clusters in the clustering $\mathcal{C}$ (also referred as the *size* of the clustering).

Our algorithm exploits the properties of a clustering of bounded radius as mentioned in the following two Lemmas.

Lemma 3. *For a given graph $G(V', E' \cup E_S)$, let $\mathcal{C}$ be a clustering with E_S as its spanning set, and let $c \in \mathcal{C}$ be a cluster having radius at-most i. For a vertex $u \notin c$, adding the least weight edge from the set $E'(u, c)$ to the spanning set E_S will ensure that the proposition $\mathcal{P}_{i+1}(e)$ holds for the entire set $E'(u, c)$.*

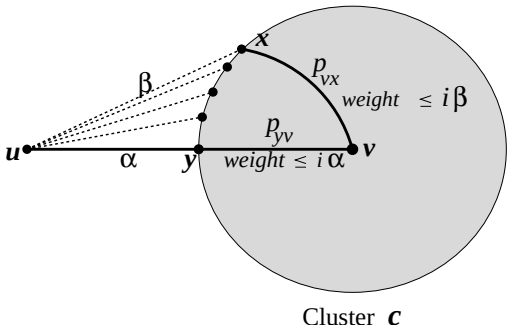

Fig. 1. Ensuring that the proposition $\mathcal{P}_{i+1}$ holds true for the set $E'(u, c)$.

Proof. Let the edge $\mathbf{e}(u,y)$ of weight α be the least-weight edge from the set $\mathcal{E}(u,c)$. Let (u,x) be any other edge of weight $\beta \geq \alpha$ from the set $\mathcal{E}(u,c)$ (see the Figure 1). Since the radius of the cluster c is at-most i, therefore, there is a path p_{xv} from x to the center v (of the cluster c) consisting of edges from the set E_S only, and its weight is at-most i times β. Using the same argument, we deduce that there is a path p_{yv} from vertex y to v consisting of edges from the set E_S, and with weight at-most $i\alpha$. Thus there is path p_{ux} from vertex u to vertex x formed by concatenating the edge (u,y) and the paths p_{yv}, p_{vx} in this order; and its weight can be bounded as follows.

$$\begin{aligned}\mathbf{w}(p_{ux}) &= \mathbf{w}(u,y) + \mathbf{w}(p_{yv}) + \mathbf{w}(p_{vx}) \\ &\leq \alpha + i\alpha + i\beta \leq \beta + i\beta + i\beta \ \{ \text{ since } \alpha \leq \beta\} \\ &= (2i+1)\beta = (2(i+1)-1)\beta\end{aligned}$$

Therefore, we can conclude that adding the edge $\mathbf{e}(u,y)$ to the spanning set makes the proposition $\mathcal{P}_{i+1}(e)$ hold true for each edge $e \in \mathcal{E}'(u,c)$.

Along similar lines we can prove the following Lemma.

Lemma 4. *For a given graph $G(V', E' \cup E_S)$, let $\mathcal{C}$ be clustering with E_S as a spanning set, and let $c_1, c_2 \in \mathcal{C}$ be two clusters having radius i and j respectively. Adding the least weight edge of the set $E'(c_1,c_2)\}$ to the spanning set E_S will ensure that the proposition $\mathcal{P}_k, k = i+j+1$ holds true for the entire set $E'(c_1,c_2)$.*

4 Algorithm for Computing a $(2k-1)$-Spanner

4.1 An Overview

Based on the key observations of a clustering of finite radius mentioned in the Lemmas 3 and 4, the algorithm for computing a $(2k-1)$-spanner of a weighted graph $G(V,E)$ selects edges to be included in the spanner in two phases.

The first phase *Forming the Clusters* starts with a clustering $\{\{v\}|v \in V\}$ (of n clusters but zero radius), and executes $\lfloor \frac{k}{2} \rfloor$ iterations. In ith iteration, a set of edges are selected to be added to the spanner that makes the proposition $\mathcal{P}_{i+1}$ true for a (possibly large) set of edge and vertices (using Lemma 3). All these edges (and vertices) are removed from the graph. A clustering is obtained again for the remaining vertices. In successive iterations, the size of the clustering (the number of clusters) reduces geometrically while the radius of clusters increase by just one unit (see Figure 2).

At the end of $\lfloor \frac{k}{2} \rfloor$ iterations, we obtain a clustering of the rest of the vertices (for whom $\mathcal{P}_{\frac{k}{2}+1}$ does not hold) that consists of only $n^{1-\frac{1}{k}\lfloor \frac{k}{2} \rfloor}$ clusters. Moreover, a subset of spanner-edges added till $\lfloor \frac{k}{2} \rfloor$th iterations spans this clustering, and ensures that the radius of each cluster is no more than $\lfloor \frac{k}{2} \rfloor$. This clustering consisting of very *few* clusters and *not-so-large* radius is passed onto the second phase of the algorithm.

In the second phase *Joining the Clusters*, the clusters are joined together by adding the least weight edge between each pair of neighboring clusters. This phase employs Lemma 4 to ensure that $\mathcal{P}_k$ holds true for all the vertices left in the graph after the first phase.

4.2 Details of the Algorithm

We describe the details of the two phases of our algorithm for computing a $(2k-1)$-spanner of a weighted graph $G(V,E)$ as follows.

Phase 1 : *Forming the clusters*

This phase executes in $\lfloor \frac{k}{2} \rfloor$ iterations. At each stage in this phase, E_S^1 denotes the set (initially $\emptyset$) of edges that are added to the spanner, and E' denotes the set of edges remaining in the graph.

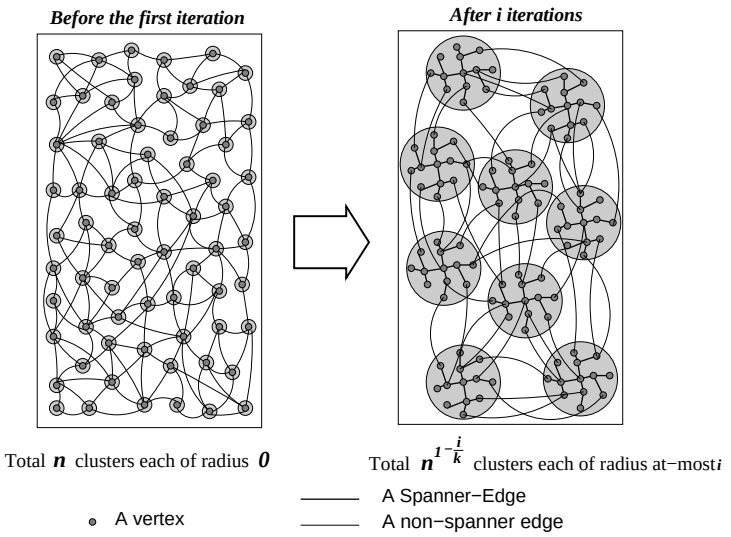

Fig. 2.

ith iteration begins with a clustering $\mathcal{C}_i$ of the set V_i' of vertices defined by the endpoints of E' after $(i-1)$th iteration.

For the first iteration, the sets are $E_S^1 = \emptyset, E' = E, V_1' = V,$, and the clustering is $\mathcal{C}_1 = \{\{v\} | v \in V\}$.

We describe the processing done in an iteration as follows.

In ith iteration, a sample $\mathcal{R}_i$ of clusters is formed by picking each cluster from the clustering $\mathcal{C}_i$ randomly independently with probability $n^{-\frac{1}{k}}$. The vertices belonging to the sampled clusters are added to the set V_{i+1}', and passed onto the next iteration as it is. Each of the remaining vertices of the set V_i' are processed according to the following two cases.

- If v is not adjacent to any sampled cluster, for each cluster $c \in \mathcal{C}_i$ adjacent to v, we add the least weight edge from the set $E'(v,c)$ to the spanner. We remove all the edges incident on v from the graph.
- If vertex v is adjacent to one or more sampled cluster, let $c \in \mathcal{R}_i$ be the cluster that is adjacent to v with edge of least weight (say w) among all the clusters from the set $\mathcal{R}_i$. We add the least weight edge from the set $E'(v,c)$ to the spanner, and remove the entire set $E'(v,c)$ from the graph. In addition, we do the following. For each cluster $c' \in \mathcal{C}_i$ adjacent to vertex v with an edge of weight $< w$, we add the least weight edge from the set $E'(v,c)$ to the spanner, and remove the entire set $E'(v,c)$ of edges from the graph.
  (It can be seen that all the edges from set E' that remain incident on the vertex v with weight $< w$ are removed in ith iteration). For the $(i+1)$th iteration, the vertex v is added to the set V_{i+1}.

For the $(i+1)$th iteration, the set V_{i+1}' thus consists of only those vertices from the set V_i' which either belong to or are adjacent to some cluster from the sample $\mathcal{R}_i$. The clustering of $\mathcal{C}_{i+1}$ for these vertices is obtained as follows. After initializing $\mathcal{C}_{i+1}$ to $\mathcal{R}_i$, each vertex $v \in V_{i+1}$ not belonging to any cluster from sample $\mathcal{R}_i$ is added to the cluster from $\mathcal{R}_i$ that is incident with the least weight edge to v. Thus a cluster $c \in C_{i+1}$ can be viewed as a union of a cluster from the sample $\mathcal{R}_i$ with the set of all those vertices from V_i' for whom c was the sampled neighboring cluster incident with the least weight edge. As a last step of ith iteration, we eliminate all the

intra-cluster edges (whose both end-points belong to the same cluster) of the clustering $\mathcal{C}_{i+1}$ from the graph (i.e., the set E').

Theorem 1. *The following assertion holds for each $j \geq 1$.*
$\mathcal{A}(j)$: For each cluster $c \in \mathcal{C}_j$, there exist a subset $\mathcal{E}$ of edges added to the spanner by the end of $(j-1)$th iteration such that c with $\mathcal{E}$ as its spanning set is a cluster of radius $\leq (j-1)$ in the graph $G(V'_j, E' \cup \mathcal{E})$.

Proof. We shall prove the theorem by induction $j \geq 1$.
Base Case : $j = 1$: In the beginning of the algorithm, the clustering is $\mathcal{C}_1 = \{\{v\} | v \in V\}$, $E_S = \emptyset, V'_1 = V, E' = E$. It is easy to observe that each cluster of $\mathcal{C}_1$ is a cluster of radius 0 in the graph $G(V, E)$ Therefore, the assertion $\mathcal{A}(1)$ holds.
Induction Hypothesis : $j \leq i$: Let the assertion $\mathcal{A}(i)$ holds.
Proof of assertion $\mathcal{A}(i+1)$:
As mentioned in the first phase of the algorithm, a cluster from the clustering $\mathcal{C}_{i+1}$ is a union $R \cup \mathcal{N}_R$ of a cluster $R \in \mathcal{R}_i$ with the set $\mathcal{N}_R$ of vertices $v \in V_i$ for whom the cluster R is the sampled cluster incident with the least weight edge among all adjacent clusters that belong to the sample $\mathcal{R}_i$. For each vertex $v \in \mathcal{N}_R$, we add to the spanner the edge (say, $\mathbf{e}(v, u), u \in R$) of least weight from the set $E'(v, R)$ in the ith iteration.

It follows from the induction hypothesis that there exists a subset $\mathcal{E}$ of edges added to the spanner by the end of $(i-1)$th iteration such that, with $\mathcal{E}$ as its spanning set, R is a cluster of radius $(i-1)$ in the graph $G(V'_i, E' \cup \mathcal{E})$. From the definition of cluster-radius, it implies that there is a vertex $r \in R$ called the center of the cluster R such that the following holds true :

For each edge $\mathbf{e}(v, x) \in E', x \in R$, there is a path from x to the vertex r, that is internal to the cluster R and consists of at-most $(i-1)$ edges from the set $\mathcal{E}$ only; and each edge on the path has weight no more than that of $\mathbf{e}(v, x)$.

Since the edge $\mathbf{e}(v, u)$ belongs to E' at the end of $(i-1)$th iteration, and is added to the spanner in ith iteration, so there is a path from the vertex v to the vertex r consisting of at-most i edges, each belonging to the spanner and having weight no more than that of the edge $\mathbf{e}(v, u)$. Also it follows from the processing of the vertex v in ith iteration that there is no edge incident on v from the set E' whose weight is less than that of $\mathbf{e}(v, u)$. Thus we can conclude that for each edge $\mathbf{e}(v, y) \in E', v \in R \cup \mathcal{N}_R$ at the end of ith iteration, there exists a subset $\mathcal{E}'$ of edges added to the spanner by the end of ith iteration so that the following holds true :

there is a path from v to the vertex $r \in R \cup \mathcal{N}_R$ (center of the cluster $R \cup \mathcal{N}_R$) that is internal to the cluster $R \cup \mathcal{N}_R$, and consists of at-most i edges, each from the set $\mathcal{E}'$, and weight of each edge being no more than that of $\mathbf{e}(v, y)$.

From definition of the cluster-radius, it follows that the cluster $R \cup \mathcal{N}_R$ with spanning set $\mathcal{E}'$ is a cluster of radius $\leq i$ in the graph $G(V'_{i+1}, E' \cup \mathcal{E}')$.

Similar arguments can be given for any other cluster in the clustering $\mathcal{C}_{i+1}$, i.e., for each cluster $c \in \mathcal{C}_{i+1}$, there is a subset $\mathcal{E}_c$ of edges added to the spanner by the end of ith iteration such that c with $\mathcal{E}_c$ as its spanning set is a cluster of radius $\leq i$ in the graph $G(V'_{i+1}, E' \cup \mathcal{E}_c)$.

Thus the assertion $\mathcal{A}_{i+1}$ holds. Hence by the principle of mathematical induction, the assertion $\mathcal{A}_j$ holds for all $j \geq 1$.

Using Lemma 3 and the Theorem given above, we can state the following corollary.

Corollary 1. *For each edge $e \in E'$ eliminated from the graph in ith iteration, the proposition $\mathcal{P}_{i+1}$ holds true.*

Since there are $\lfloor \frac{k}{2} \rfloor$ iterations in the first phase, it follows from the corollary stated above that $\mathcal{P}_k$ holds for each edge eliminated from the graph in the first phase.

We shall now bound the expected number of edges added to the spanner in the first phase.

Lemma 5. *The expected number of edges added to the spanner in each iteration of the first phase of our algorithm is $n^{1+1/k}$.*

Proof. Consider ith iteration, and a vertex $v \in V_i'$. All the neighbors of the vertex v are grouped into their respective clusters of the clustering $\mathcal{C}_i$. Let $c_1, c_2, \cdots, c_l$ be the clusters adjacent to v, and arranged in the increasing order of the weight of their least-weight edge incident on v, i.e., the least weight edge from a set $E'(v, c_j)$ is lighter (has less weight) than the least weight edge from the set $E'(c, c_{j+1})$ for all $j < l$.

It follows from the algorithm that for the cluster c_j adjacent to v, we add just one edge (the least weight edge) from the set $E'(v, c_j)$ to the spanner, if none of the clusters preceding it, i.e., $c_1, \cdots, c_{j-1}$ are sampled. Since each cluster is sampled independently with probability $n^{-1/k}$, the probability that we add an edge from $E'(v, c_j)$ to the spanner is no more than $(1 - n^{-1/k})^{j-1}$. Thus the expected number of edges contributed to the spanner by a vertex $v \in V_i$ is given by

$$\sum_{j=1}^{j=l} \left(1 - n^{-1/k}\right)^{j-1} \leq \frac{1}{n^{-1/k}} = n^{1/k}$$

Thus the expected number of edges added to the spanner in ith iteration is bounded by $n^{1+1/k}$.

We repeat an iteration if the number of edges exceed $O(n^{1+1/k})$ - the expected number of repetitions is $O(1)$. There are total $\lfloor \frac{k}{2} \rfloor$ iterations, so the expected number of edges added to the spanner in the first phase is $O(kn^{1+1/k})$.

Let E' be the set of edges left in the graph after first phase, and let V' be the end-points of the edges E'. The first phase outputs a clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ for the vertices V'. Note that after each iteration in the first phase, the number of clusters reduces by a factor of $n^{1/k}$. Since there are $\lfloor \frac{k}{2} \rfloor$ number of iterations, the clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ has only $n^{1-\frac{1}{k}\lfloor \frac{k}{2} \rfloor}$ clusters. Moreover, it follows from the Theorem 1 that a subset of edges added to the spanner in first phase of the algorithm spans the clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ and ensures a bound of $\lfloor \frac{k}{2} \rfloor$ on the radius of each cluster $c \in \mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$.

In order to ensure that the proposition $\mathcal{P}_k$ holds for the remaining vertices V' also, the algorithm makes use of Lemma 4 in the second phase of addition of edges.

Phase 2 : *Joining the Clusters*

In this phase, we perform the following two operations of adding the edges to the spanner depending on whether k is even or odd.

- If k is odd, then for each pair of clusters $c', c'' \in \mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$, we add the least-weight edge between the two clusters (i.e., the least weight edge from the set $E'(C(u), C(v))$) to the spanner E_S. It follows from Lemma 4 that $\mathcal{P}_k$ holds for each edge $e \in E'$. Also note that the number of edges added is at-most square of the number of clusters, i.e., $|\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}|^2$, which is $n^{1+1/k}$.
- If k is even, then for each cluster $c \in \mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$, we do the following. We group the vertices incident on the cluster c in their respective clusters of the clustering at the end of the $(\lfloor \frac{k}{2} \rfloor - 1)$th iteration (the second last iteration). For each group of vertices (belonging to same cluster in the second last iteration) we pick the least-weight edge among the set of edges incident from these vertices to the cluster c, and add it to the spanner. It follows from Lemma 4 that $\mathcal{P}_k$ holds for each edge $e \in E'$.
  The number of edges added is at-most the number of clusters in the last clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ times the number of clusters in the second last clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor - 1}$, which is $n^{1+1/k}$.

Figure 3 shows how the clusters are joined in the second phase of our algorithm for building the spanner of stretch 3,5,7 and 9 (i.e., $k = 2, 3, 4, 5$).

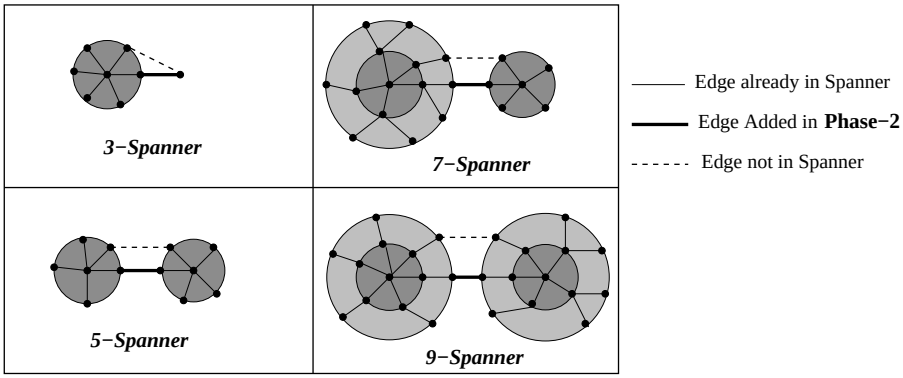

Fig. 3. Joining the clusters in the second phase, and ensuring the stretch-bound

Let E_S^2 be the set of edges added to the spanner in the second phase of our algorithm. From the Theorem 1, and the arguments given above, we can state the following theorem.

Theorem 2. *If E_S^1, and E_S^2 are the sets of edges selected to form the spanner in the first and the second phase of our algorithm for the graph $G(V, E)$ and parameter k, then the sub-graph $G(V, E_S^1 \cup E_S^2)$ is a $(2k-1)$-spanner for the graph $G(V, E)$ and consists of $O(kn^{1+1/k})$ edges.*

Running time of the Algorithm :

We shall now show that both the phases of our algorithm run in $O(km)$ time. For sake of brevity, we sketch the analysis of running time of the first phase only (the analysis for the second phase is analogous).

It is easy to observe that having sampled a subset $\mathcal{R}_i$ of clusters from $\mathcal{C}_i$, it will take a total of $O(|E'|)$ time to find out the neighboring cluster (if exists) for each vertex $v \in V_i'$ that is incident with the least weight edge among all the sampled clusters adjacent to v.

In order to perform the remaining task of the ith iteration, we need an efficient way to pick the least weight edge from a set $E'(v, c), c \in \mathcal{C}_i$ (to be added to the spanner), followed by removing the entire set $E'(v, c)$ from E'. This entire task of ith iteration can be accomplished in $O(|E'|)$ time if adjacency list of each $v \in V_i'$ is ordered in such a way that the edges incident on v from the vertices belonging to same cluster appear contiguous in the adjacency list of v. Such an order of edges in the adjacency lists can be achieved in $O(|E'|)$ time using a radix sort on the end-points of the edges as follows.

A clustering $\mathcal{C}$ on a set of vertices V' can be expressed by a labeling function $f_C : V' \rightarrow V'$ where $f_C(u) = f_C(v)$ iff both u and v belong to the same clusters in the clustering $\mathcal{C}$. Given a clustering $\mathcal{C}$, such a labeling f_C can be defined by labeling all the vertices of a cluster by a vertex (arbitrary one) picked from the same cluster. Given a graph $G(V, E)$ with a clustering $\mathcal{C}$ and its associated function f_C, we do the following :

We concatenate all the adjacency lists of the vertices to form a list $\mathcal{L}$. Note that an edge between a vertex u and a vertex v appears twice in this list - once as $\mathbf{e}(u, v)$ (from the adjacency list of u), and once as $\mathbf{e}(v, u)$ (from the adjacency list of v). First we sort $\mathcal{L}$ on the label of the second end-point of edges as defined by f_C. This will bring together all the edges whose second end-points belongs to the same cluster. Let $\mathcal{L}'$ be the new list after the sorting. Now we sort $\mathcal{L}'$ on the first end-point of edges. This will arrange all the edges of $\mathcal{L}'$ in such a way that edges belonging to adjacency list of a vertex appear together, and within each adjacency list also the edges incident from same cluster appear together. The entire process of getting this desired ordering as explained above takes $O(|E'|)$ time since the radix sort runs in linear time.

Thus each iteration of the first phase of our algorithm runs in $O(|E'|) = O(m)$ time. Total number of iterations being $\lfloor \frac{k}{2} \rfloor$, it can be concluded that the running time of the first phase of our algorithm is $O(km)$. We repeat an iteration of the first phase if the total number of edges exceed $O(n^{1+1/k})$, and the expected number of repetitions is $O(1)$. Along similar lines, it can be shown that the second phase of our algorithm runs in $O(m)$ time. Thus the expected running time of our algorithm is $O(km)$. Combining with Theorem 2, we state the following Theorem.

Theorem 3. *Given a weighted graph $G(V, E)$, and integer $k > 1$, a spanner of stretch $(2k-1)$ and $O(n^{1+1/k})$ size can be computed in expected time $O(km)$.*

References

1. I. Althofer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete and Computational Geometry*, 9:81–100, 1993.
2. B. Awerbuch. Complexity of network synchronization. *Journal of Ass. Compt. Mach.*, pages 804–823, 1985.
3. H. J. Bandelt and A. W. M. Dress. Reconstructing the shape of a tree from observed dissimilarity data. *Journal of Advances of Applied Mathematics*, 7:309–343, 1986.
4. B. Bollobas. In *Extremal Graph Theory*, page 164. Academic Press, 1978.
5. J.A. Bondy and M. Simonovits. Cycle of even length in graphs. *Journal of Combinatorial Theory, Series B*, 16:97–105, 1974.
6. Edith Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM J. Comput.*, 28:210–236, 1998.
7. P. Erdos. Extremal problems in graph theory. In *In Theory of Graphs and its Applications(Proc. Sympos. Smolenice,1963)*, pages 29–36. Publ. House Czechoslovak Acad. Sci., Prague,1964, 1963.
8. Shay Halperin and Uri Zwick. Unpublished result. 1996.
9. David Peleg and A. A. Schaffer. Graph spanners. *Journal of Graph Theory*, 13:99–116, 1989.
10. David Peleg and Eli Upfal. A trade-off between space amd efficiency for routing tables. *Journal of Assoc. Comp. Mach.*, 36(3):510–530, 1989.
11. Mikkel Thorup and Uri Zwick. Approximate distance oracle. In *Proceedings of 33rd ACM Symposium on Theory of Computing (STOC)*, pages 183–192, 2001.
12. Uri Zwick. Personal communication.