A Simple Linear Time Algorithm for Computing Sparse Spanners in Weighted Graphs *

Surender Baswana† and Sandeep Sen‡

Department of Computer Science and Engineering,
Indian Institute of Technology 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 computes a $(2k-1)$-spanner of size matching the conjectured lower bound.

An important feature of our algorithm is its local approach. Unlike all the previous algorithms which require computation of shortest paths, the new algorithm merely explores the edges in the neighborhood of a vertex or a group of vertices. This feature leads to designing simple external-memory and parallel algorithms for computing sparse spanners, whose running times are optimal up to logarithmic factors.

Keywords: Graph algorithms, Randomized algorithms, Shortest path

---

*Preliminary version of this work appeared in 30th International Colloquium on Automata, Languages and Programming (ICALP), pages 384-396, 2003.

†The work was supported in parts by a fellowship from Infosys Technologies Ltd., Bangalore.

‡The work was supported in parts by an IBM UPP award.

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 called the *stretch factor* associated with the t-spanner.

There are many applications in distributed systems and communication networks where spanners appear as an underlying graph structures. For example, in order to maintain compact routing tables [11], many existing routing schemes use the edges of a sparse spanner for routing messages. Peleg and Ullman [2, 10] show that the sparseness and stretch of a spanner of a given undirected graph is very closely related to the communication complexity of any synchronizer for the network based on the graph.

Efficiently computable spanners are also used for designing efficient algorithms for computing all-pairs approximate shortest paths in dense graphs. The running time of algorithms for all-pairs shortest paths (APSP) is proportional to the number of edge. Therefore, computing a sparse t-spanner of a given dense graph followed by running any APSP algorithm on the spanner gives all-pairs t-approximate shortest paths. For these applications, it is essential that the task of computing t-spanner takes much less time than that of the algorithm for computing APSP.

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, 11].

All the applications of spanners require a t-spanner of smallest possible size. Therefore, from graph theoretic perspective, the following question arises : How sparse can a t-spanner of a graph be? In this regard, a lot of work [9, 14] has been done to establish a lower bound on the size of spanner in terms of stretch factor. These existing results use the following simpler 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 $n^{1+1/k}$ 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 has 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 $2k + 2$. These graphs can't have any t-spanner for $t < 2k + 1$, except the graph itself. Thus there is a lower bound of $O(n^{1+1/k})$ on the size of t-spanner for $t \leq 2k$.

Althöfer et al. [1] gave the first algorithm for computing a $(2k-1)$-spanners for weighted graphs, and the size of the spanner is $O(n^{1+1/k})$. The lower bound mentioned above implies that $O(n^{1+1/k})$ bound on the size of $(2k-1)$-spanner is essentially optimal. However, the best known implementation of the algorithm of Althöfer et al. [1] for $(2k-1)$-spanner has a running time of $O(mn^{1+1/k})$. Cohen [6] gave a randomized algorithm with $O(kmn^{1/k})$ expected running time for computing a spanner of size $O(kn^{1+1/k})$ and slightly larger stretch $(2k+\epsilon)$. Thorup and Zwick [14], improving the result of Cohen [6] gave a randomized algorithm for computing a $(2k-1)$-spanner of optimal size in $O(kmn^{1/k})$ expected time. All these existing algorithms for computing spanners in undirected weighted graphs 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 [14]. 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 $O(m)$.

Improved spanner-results exists for special classes of graphs : chordal graphs, unweighted graphs, Euclidean graphs in d-dimensions. For chordal graphs, Peleg and Schaffer [9] show a lower bound of $O(n^{3/2})$ on the size of 2-spanner and give an algorithm for computing 2-spanner with size matching this lower bound. They also present an algorithm for computing a 3-spanner of $O(n \log n)$ size for chordal graphs. For unweighted graphs, Halperin and Zwick [8, 15] gave an $O(m)$ time algorithm to construct a $(2k-1)$-spanner of size $O(n^{1+1/k})$. Salowe [13] present an algorithm for computing $(1+\epsilon)$-spanner of a d-dimensional complete Euclidean graph in $O(n \log n + \frac{n}{\epsilon^d})$ time. However, none of the existing algorithms for these special classes of graphs seem to be extensible to general weighted undirected graphs.

1.1 Our Contribution

We present a randomized algorithm that takes expected linear time for computing $(2k-1)$-spanner of optimal size for a given weighted undirected graph. 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 the 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 parallel computation environment and in external memory as well. For CRCW PRAM model, we present a parallel algorithm that computes a $(2k-1)$-spanner in $O(k)$ steps only while performing $O(mk)$ work. For EREW PRAM model, the running time and the total work of the parallel algorithm degrades by just a logarithmic factor. We also present an external-memory algorithm for computing a $(2k-1)$-spanner that has same running time as that of sorting m integers in external-memory.

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. In the following section, as a warm-up, we present an $O(m)$ expected time algorithm for computing a 3-spanner, and mention some of the key ideas (clustering of vertices) which we formalize and extend in section 4. We present our linear time sequential algorithm for computing $(2k-1)$-spanner in section 5. We outline the parallel and external-memory algorithms for computing sparse spanners in section 6. Throughout the paper, we assume that the input graph is given in the form of adjacency list. Without loss of generality, it is also assumed that the edge weights are distinct.

2 Computing a 3-Spanner

In order to compute a 3-spanner of a given weighted graph $G(V, E)$ with potentially $\theta(n^2)$ edges, the objective is to select at most $O(n^{3/2})$ edges to be included in the spanner, and still ensure that the distance between any pair of vertices is stretched no more than three times in the spanner. The algorithm selects edges to be included in the spanner in two phases.

1. *Forming the clusters* :
   We choose a sample $\mathcal{R} \subset V$ by picking each vertex independently with probability $n^{-\frac{1}{2}}$. We form clusters (of vertices) around the sampled vertices. 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 each unsampled vertex $v \in V - \mathcal{R}$ as follows.

   - If v is not adjacent to any sampled vertex, we add every edge incident on v 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 $\mathbf{e}(v, \mathcal{N}(v, \mathcal{R}))$, and every other edge incident on v with weight less than that of $e(v, \mathcal{N}(v, \mathcal{R}))$ to the spanner. The vertex v is added to the cluster centered at $\mathcal{N}(v, \mathcal{R})$.

   At the end of the first phase, let E' denote the set of all the edges of the given graph excluding the edges added to the spanner and all the intra-cluster edges. Let V' be the set of vertices corresponding to the end-points of the edges E'. It should be noted that each vertex of set V' is either a sampled vertex or is adjacent to some sampled vertex, and the first phase has partitioned V' into disjoint clusters each centered around some sampled vertex. The graph $G(V', E')$ is passed onto the second phase.

2. *Joining the Clusters*:
   For each vertex $v \in V'$, 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 select

the least-weight edge among all the edges between (the vertices of) the cluster and v, and add it to the spanner.

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 2.1 *The weight of each edge* $\mathbf{e}(u, v) \in E - E_S^1$, *is greater or equal to the weight of the edge between* v *and* $\mathcal{N}(v, \mathcal{R})$ *(the center of the cluster to which* v *belongs)*

Lemma 2.2 *For an edge* $\mathbf{e}(u, v) \in E$ *that is not present in the spanner* $G(V, (E_S^1 \cup E_S^2)$, *the following assertion holds true:*
There is a path between u *and* v *in the spanner consisting of at-most three edges, each of weight no more than that of the edge* $\mathbf{e}(u, v)$.

Proof: It follows from the first phase of the algorithm that u as well as v is adjacent to one or more vertices of the sample $\mathcal{R}$, and therefore belong to same/different clusters. There are two cases now.
Case 1 : Both the vertices belong to same cluster, say the cluster centered at $x \in \mathcal{R}$. It

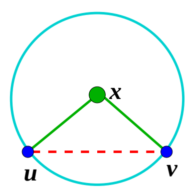


Figure 1: vertex u belongs to the same cluster as the vertex v

follows from Lemma 2.1 that there is a 2-edge long path $u - x - v$ in the spanner with weight no more than twice the weight of the edge $\mathbf{e}(u, v)$. This provides a justification for not processing any intra-cluster edge in the second phase (an intra-cluster edge needs not be added to the spanner).
Case 2 : The vertices u and v belong to different clusters, and let v belong to the cluster centered at $x \in \mathcal{R}$ (see figure below).

Let $\mathbf{e}(u, v')$ be the least weight edge $\in E - E_S^1$ among all the edges incident on u from the vertices of the cluster centered at x. It follows from the second phase of our algorithm, that the edge $\mathbf{e}(u, v')$ gets added to the spanner. Hence there is a path $p_{uv} = u - v' - x - v$ between u and v in the spanner and its weight can be bounded as follows.

$$\begin{aligned}
weight(p_{uv}) &= weight(\mathbf{e}(u, v')) + weight(\mathbf{e}(v', x)) + weight(\mathbf{e}(x, v)) \\
&\leq weight(\mathbf{e}(u, v')) + weight(\mathbf{e}(u, v')) + weight(\mathbf{e}(x, v)) \ \ \{\text{using Lemma 2.1 }\} \\
&\leq 2weight(\mathbf{e}(u, v')) + weight(\mathbf{e}(u, v)) \ \ \{\text{using Lemma 2.1 }\} \\
&\leq 3weight(\mathbf{e}(u, v)) \ \ \{\text{follows from the second phase of the algorithm}\}
\end{aligned}$$

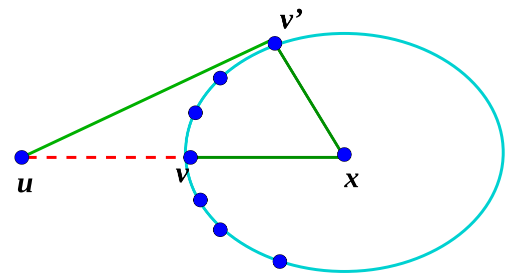


Figure 2: vertex u does not belong to the cluster containing vertex v

□

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 2.2). 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 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 size of the set E_S^1 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 $O(n^{3/2})$.

Lemma 2.3 *Both the phases of the algorithm for computing a 3-spanner can be implemented in $O(m)$ time.*

Proof: Without loss of generality we assume that the vertices are numbered 1 to n. The random sample $\mathcal{R}$ can be chosen in $O(n)$ time. Also the nearest sampled neighbor for a vertex $v \in V$ can be computed by a single traversal of the adjacency list of the vertex v. Let N be the array storing nearest sampled neighbor for each vertex (if exists). The remaining task of the phase-1 is to select (and add to the spanner), for each vertex v, all the edges incident with weight less than that of the edge $\mathbf{e}(v, \mathcal{N}(v, \mathcal{R}))$. This can be performed by traversing adjacency list of each vertex.

In the second phase of the algorithm, for each vertex v we have to select the least weight edge from each cluster incident on v. For this purpose, we use an auxiliary array $A[1..n]$ whose entries point to *null* initially. We scan the adjacency list of a vertex v, and process an edge $\mathbf{e}(v,w)$ as follows. Let $x \in \mathcal{R}$ be the center of the cluster to which the vertex w belongs (note that the center of the cluster to which vertex w belongs can be accessed in constant time by reading $N[w]$). We make $A[x]$ point to the edge $\mathbf{e}(v,w)$ if it is pointing to *null* previously or if the weight of the edge $\mathbf{e}(v,w)$ is less than that of pointed by $A[x]$. It is easy to verify that once all the edges incident on v have been processed, an entry $A[x]$ (if not *null*) points to the least weight edge incident on v among all the edges incident from the cluster centered at $x \in \mathcal{R}$. Now we perform another scan of the adjacency list of vertex v. For each edge $\mathbf{e}(v,w)$, if $A[w]$ is not *null*, we add the edge pointed by $A[w]$ to the spanner, and make $A[w]$ point to *null*. It can be seen that in this way, just by two traversals of the adjacency list of the vertex v, we can select (and add to the spanner) the least-weight edge incident on v from each neighboring clusters of v, and also the array A gets restored to initial state (all its entries pointing to *null*) to be used for another vertex.

Thus with an extra space (arrays A and N) of size n, both the phases of the algorithm for computing a 3-spanner can be implemented in $O(m)$ time.

□

We can thus conclude that for a given weighted undirected graph, a 3-spanner of size $O(n^{3/2})$ can be computed in $O(m)$ expected time.

3 Key Ideas Underlying the Algorithm

Given a weighted graph $G(V,E)$ and a subset $E_S \subset E$, consider the following proposition $\mathcal{P}_i(e)$ defined for an edge $e \in E - E_S$ and an integer i:

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

We say that $\mathcal{P}_i(v)$ holds for a vertex v if $\mathcal{P}_i(e)$ holds for each edge e incident on the vertex v.

It can be seen that the task of computing a $(2k-1)$-spanner for a graph $G(V,E)$ amounts to finding a subset $E_S \subset E$ such that $\mathcal{P}_{2k-1}(e)$ holds for each edge $e \in E - E_S$. Now in order to pick such a set E_S of $O(n^{1+1/k})$ edges from potentially $\theta(n^2)$ edges in a graph, the key idea underlying our algorithm is the partitioning of the 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 compared to the number of vertices 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 3.1 *A* **cluster** *is a subset of vertices. A partition of a set $V' \subset V$ into clusters is called* **clustering** *of V'.*

As will soon become clear in the context of our algorithm, each cluster is a singleton set in the beginning, and other vertices gets added to the cluster as the algorithm proceeds. We shall denote this (unique) oldest member of a cluster as the **center of the cluster**.

A clustering $\mathcal{C}$ can be defined by a function $f_C : V \to V$ such that $f_C(u)$ is the center of the cluster to which the vertex u belongs. Note that $f_C(u) = f_C(v)$ iff vertices u and v belong to same cluster. Hence the function f_C associated with the clustering $\mathcal{C}$ can be used to determine whether two vertices belong to the same cluster or not.

Definition 3.2 *A set $E_S \subset E$ 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 3.3 *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 the following holds true :*

For each edge $\mathbf{e}(x, y) \in E' - E_S, x \in c, y \notin c$, there is a path from x to the center of the cluster which is internal to the cluster and consists of at-most i edges, each from the set E_S and having weight no more than that 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 3.4 *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.

- $E(x, c)$: the edges from the set E that are incident from the vertices of cluster c to the vertex x.
- $E(c_1, c_2)$: the set of edges between vertices of cluster c_1 and vertices of cluster c_2 that belong to the set E.
- $\min(E')$: the least weight edge from the set E'.

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

Lemma 3.1 *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 $\mathcal{P}_{2i+1}(e)$ holds for each edge $e \in E'(u, c)$.*

Proof: Let the edge $\mathbf{e}(u, y)$ of weight α be the least-weight edge from the set $E(u, c)$. Let

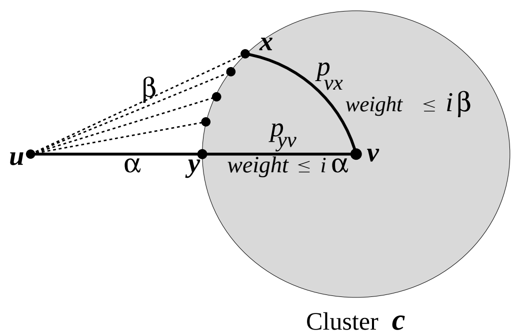


Figure 3: Ensuring that the proposition $\mathcal{P}_{2i+1}$ holds for the set $E'(u, c)$.

(u, x) be any other edge of weight $\beta \geq \alpha$ from the set $E(u, c)$ (see the Figure 3). 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.

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

Therefore, we can conclude that adding the edge $\mathbf{e}(u, y)$ to the spanning set makes the proposition $\mathcal{P}_{2i+1}(e)$ hold true for each edge $e \in \mathcal{E}'(u, c)$. $\square$

Along similar lines we can prove the following Lemma.

Lemma 3.2 *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 $\mathcal{P}_{2i+2j+1}$ 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 3.1 and 3.2, the algorithm for computing a $(2k-1)$-spanner of a weighted graph $G(V, E)$ selects the 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 and 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 $\mathcal{P}_{2i-1}$ true for a possibly large set of edges (and vertices) using Lemma 3.1. 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, i.e., the number of clusters reduces geometrically while the radius of clusters increase by just one unit.

At the end of $\lfloor \frac{k}{2} \rfloor$ iterations, we obtain a clustering 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 iteration 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 3.2 to ensure that $\mathcal{P}_{2k-1}$ holds 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 denotes the set of edges that are added to the spanner, and E' denotes the set of edges remaining in the graph. Let V'_i denote the set of vertices defined by the end-points of E' after $(i-1)$th iteration.

For the first iteration, the sets are $E_S = \emptyset, E' = E, V_1' = V$. We describe the processing done in an iteration as follows.

The ith iteration begins with a clustering $\mathcal{C}_i$ of the set V_i'. (For the first iteration $\mathcal{C}_1 = \{\{v\}|v \in V\}$), and performs the following steps in order.

1. A sample $\mathcal{R}_i$ of clusters is chosen by picking each cluster from the clustering $\mathcal{C}_i$ 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. The clustering $\mathcal{C}_{i+1}$ is initialized to $\mathcal{R}_i$.

2. Each vertex $v \in V_i'$ not belonging to any sampled cluster is 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 the clustering $\mathcal{C}_{i+1}$, the vertex v is grouped with the cluster c. 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 $< w$ get removed from E' 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 $\mathcal{R}_i$. Also the clustering $\mathcal{C}_{i+1}$ for the set V_{i+1} is such that a cluster $c \in C_{i+1}$ can be viewed as a union of a sampled cluster $R \in \mathcal{R}_i$ with the set of all those vertices from V_i' for whom R was the sampled neighboring cluster incident with the least weight edge.

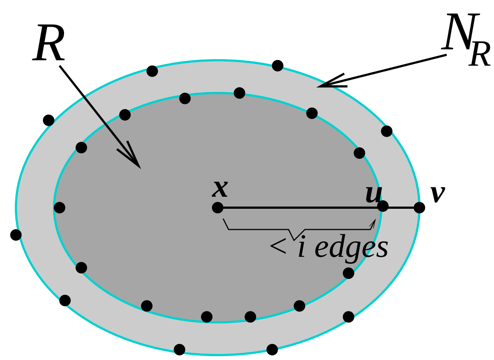
3. All the intra-cluster edges (whose both end-points belong to the same cluster) of the clustering $\mathcal{C}_{i+1}$ are eliminated from the graph (i.e., the set E').

Observation 4.1 *A cluster $c \in \mathcal{C}_i$ is selected for the next iteration with probability $n^{-\frac{1}{k}}$. Therefore, the expected number of the clusters in the clustering $\mathcal{C}_i$ is $n^{1-\frac{i}{k}}$.*

We shall now show that $\mathcal{P}_k$ holds for each edge eliminated from the set E in phase-1. In this regard, the following theorem is crucial.

Theorem 4.1 *The following assertion holds for each $j \geq 1$.*
$\mathcal{A}(j)$ *: For each cluster $c \in \mathcal{C}_j$, there exist a subset $\mathcal{E} \subset E_S$ of spanner edges 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 on $j \geq 1$.
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 $\mathcal{C}_1$ with $\emptyset$ as a spanning set 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 observed at the end of step *2* of the ith iteration, 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 all those vertices of set V_i for whom the cluster R is the nearest sampled neighboring cluster. It follows from Definition 3.1 that the center of the cluster $R \cup \mathcal{N}_R$ is same as that of the cluster R. 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 $G(V'_i, E' \cup \mathcal{E})$. I.e., for each edge $\mathbf{e}(v, x) \in E', x \in R$, there is a path from x to the center of the cluster, that is internal to the cluster R and consists of at-most $(i-1)$ edges from the set $\mathcal{E}$ each of weight no more than that of $\mathbf{e}(v, x)$. Consider a vertex $v \in \mathcal{N}_R$. During ith iteration, we add to the spanner the edge (say, $\mathbf{e}(v, u), u \in R$) of least weight from the set $E'(v, R)$. Therefore, there is a path consisting of at-most i spanner edges from v to the center of the cluster $R \cup \mathcal{N}_R$, and the weight of each edge on this path is no more than that of $\mathbf{e}(v, u)$. Also it is observed from step *2* of the 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 is a path from v to the center of the cluster consisting of at-most i spanner-edges each of weight no more than that of $\mathbf{e}(v, y)$. In other words, the cluster $R \cup \mathcal{N}_R \in \mathcal{C}_{i+1}$ with spanning set $\mathcal{E}' = \mathcal{E} \cup \{\min(E'(v, R)) | v \in \mathcal{N}_R\}$ 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 $\mathcal{C}_{i+1}$. Thus the assertion $\mathcal{A}_{i+1}$ holds. Hence by the principle of mathematical induction, the assertion $\mathcal{A}_j$ holds for all $j \geq 1$. $\square$

Using Lemma 3.1 and the Theorem given above, we can state the following Theorem.

Theorem 4.2 *For each edge $e \in E'$ eliminated from the graph in the first phase, the proposition $\mathcal{P}_k$ holds.*

Proof: Let $\mathbf{e}(u, v)$ be an edge eliminated from E' during ith iteration. Note that the edges are eliminated from the set E' only in the second or the third step of ith iteration.

Case 1 : (*The edge $\mathbf{e}(u, v)$ is eliminated from E' during the second step*)

Note that we surely add the least weight edge from u and the cluster to which the vertex v belongs. It follows from Theorem 4.1 that each cluster during ith iteration has radius $i-1$. Therefore, using Lemma 3.1 the proposition $\mathcal{P}_{2i-1}$ holds for the edge $\mathbf{e}(u, v)$.

Case 2: (*The edge $\mathbf{e}(u, v)$ is eliminated during fourth step*).

In this case both u and v must be assigned to same cluster say $c \in \mathcal{R}_i$. Also it follows from the step *2* of the ith iteration that the edge $\mathbf{e}(u, v)$ is heavier than the edge $\min(E'(u, c))$ that we add to the spanner. It follows from Theorem 4.1 that c is a cluster of radius $i - 1$. Therefore, there is a path from u to the center of the cluster c consisting of at-most i spanner-edges, each of weight no more than that of $\mathbf{e}(u, v)$. Similarly, there is a path from v to the center of the cluster c consisting of at-most i spanner-edges, each of weight no more than that of $\mathbf{e}(u, v)$. Hence there is a path from u to v passing through the center of the cluster and consisting of at-most $2i$ spanner-edges, each of weight less than that of $\mathbf{e}(u, v)$. In other words $\mathcal{P}_{2i}$ holds for the edge $\mathbf{e}(u, v)$.

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

Lemma 4.1 *The expected number of edges added to the spanner by the first phase is $kn^{1+1/k}$, and expected running time of the first phase is $O(km)$.*

Proof: Consider ith iteration of the first phase, 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 total number of edges added to the spanner in the first phase is $O(kn^{1+1/k})$.

We now address the running time of the first phase of the algorithm. An iteration of this phase begins with choosing a randoms sample of clusters and finding the neighboring sampled cluster nearest to each vertex. It is easy to perform these steps in $O(|E'|)$ time. The remaining tasks of the iteration (selecting $\min(E'(v,c))$ and/or eliminating $E'(v,c)$) are similar to those of the second phase of the algorithm for computing a 3-spanner, and thus can be implemented in $O(|E'|)$ time using an extra $O(n)$ size space as follows from Lemma 2.3. Thus the running time of an iteration is $O(|E'|) = O(m)$. As mentioned above, an iteration gets repeated only a constant expected number of times. Since there are total $\lfloor \frac{k}{2} \rfloor$ iterations in the first phase, therefore, the expected running time of the first phase is $O(km)$. $\square$

At the end of the first phase of the algorithm, let E_S be the set of edges added to the spanner and E' be the set of edges still left in the graph. The vertices V' corresponding to the end-points of E' are partitioned in the clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ that consists of $n^{1-\frac{1}{k}\lfloor \frac{k}{2} \rfloor}$ clusters (using Observation 4.1). Moreover, the Theorem 4.1 implies that $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ with E_S as its spanning set is a clustering of radius $\lfloor \frac{k}{2} \rfloor$.

The graph $G(V', E')$ with the clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ is passed on to the second phase of the algorithm.

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 $E'(c, c')$) to the spanner E_S.

- 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.

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

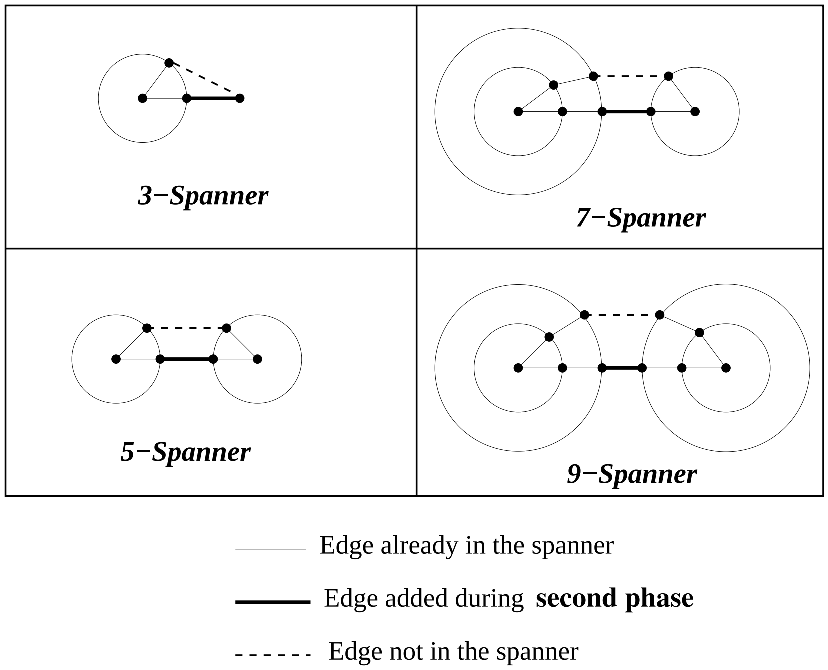


Figure 4: Joining the clusters in the second phase, and ensuring the stretch-bound

Theorem 4.1 implies that $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ and $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor - 1}$ are clusterings of radius $\lfloor \frac{k}{2} \rfloor$ and $\lfloor \frac{k}{2} \rfloor - 1$ respectively. Therefore it follows from Lemma 3.2 that the proposition $\mathcal{P}_{2k-1}$ holds for each edge of the graph at the end of the second phase.

Lemma 4.2 *The total number of edges added to the spanner by the second phase is* $n^{1+1/k}$.

Proof: We handle the odd-even case of k separately.

k is odd : 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 $n^{1+1/k}$ using Observation 4.1.

k is even : The number of edges added is at-most the number of clusters in the clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ times the number of clusters in the clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor - 1}$. So the total number of edges added to the spanner in the second phase is $n^{1+1/k}$ using Observation 4.1.

□

It is easy to observe that just like an iteration of the first phase, the second phase of the algorithm has a running time of $O(m)$. We can now conclude the following theorem.

Theorem 4.3 *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* $O(km)$*.*

5 Efficient Implementation of the Algorithm in Various Programming Environments

An important aspect that distinguishes our algorithm from the previous algorithms for computing sparse spanners is its local approach. At any step, the algorithm manipulates the edges in the neighborhood of a vertex or a cluster. Due to its local approach, it is possible to design near optimal external-memory algorithm and parallel algorithm for computing sparse spanners.

We shall first show that a suitable implementation of our algorithm in external memory has running time same as that of sorting m integers in external memory. Thereafter we shall show that an implementation of our algorithm in parallel CRCW environment runs in $O(k)$ steps and performs optimal work $O(km)$.

For describing external-memory algorithm (and parallel algorithm) for computing a $(2k-1)$-spanner, essentially all we need to show is efficient implementation of the following three main steps in external memory (and parallel)

1. Sampling a subset of clusters from a given set of clusters
2. Computing nearest sampled neighboring cluster for each vertex if exists.
3. Finding the least weight edge from the set $E(v,c)$ of edges between a vertex v and a cluster c.

For each implementation, we assume that the initial graph is given in form of adjacency list. Note that an edge between a vertex u and a vertex v appears twice in the adjacency list representation - once as $\mathbf{e}(u,v)$ (in the adjacency list of u), and once as $\mathbf{e}(v,u)$ (in the adjacency list of v).

5.1 Implementation in external-memory

We assume the most widely accepted external-memory model. Let B be the block size and μ be the size of internal-memory in blocks. We shall show that with the set of edges E' arranged in some suitable order, each of the three tasks mentioned above an be performed in I/O optimal way.

The function f_C corresponding to a clustering C is represented as an array. We assume that in addition to storing both the end-points, and weight, the data-structure for each edge $\mathbf{e}(u,v)$ also stores the label of the clusters to which its end-points belong.

Given two clustering $\mathcal{C}, \mathcal{C}'$ in the form of the arrays of their labeling functions $f_C, f_{C'}$, let $Sort(E', f_C, f_{C'})$ be the procedure that arranges the set of edges E' in a list $L(E')$ so that for any two clusters $c \in \mathcal{C}, c \in \mathcal{C}'$,

1. the set of edges $\{\mathbf{e}(u,v)|u \in c\}$, i.e. the edges emanating from the cluster c appear as a sub-list say $L_c(E')$ in the list $L(E')$,

2. The set of edges $E'(c,c')$ appear as a sub-list within the sub-list $L_c(E')$

The procedure $Sort(E', f_C, f_{C'})$ can be implemented using integer-sort on the labels of the the end-points of the edges, thus has a running time of $O(|E'|)$ in RAM model. However in external-memory, integer sorting has same lower-bound and upper-bound in the running time as that of generic sorting problem, which is $O(\frac{|E|}{B}\log_\mu \frac{|E|}{B})$.

The three basic tasks for computing $(2k-1)$-spanner can be performed in external memory as follows.

- In order to form a sample of clusters, we perform $Sort(E', f_{\mathcal{C}_i}, f_{\mathcal{C}_1})$. As a result, the adjacency lists of all the vertices belonging to a cluster of the clustering $\mathcal{C}_i$ appear contiguous as a sub-list. To form a sample R of clusters, we scan this list of edges E', picking each cluster for the sample independently with probability $n^{-1/k}$, and forming a list $E'(R)$ of edges emerging from the sample R of clusters.

- In order to compute the nearest sampled neighboring cluster for each vertex, if exists, we perform $Sort(E'(R), f_{\mathcal{C}_1}, f_{\mathcal{C}_i})$. As a result, the edges incident on a vertex from all the sampled clusters appear contiguous. It is easy to observe that merely a linear scan of this edge-list is enough to compute the nearest sampled neighboring cluster for all the vertices.

- Finally we rearrange the edges E' so that the edges incident on a vertex from a given cluster of the clustering $\mathcal{C}_i$ appear contiguous. This requires $Sort(E', f_{\mathcal{C}_1}, f_{\mathcal{C}_i})$. This rearrangement ensures that all the remaining tasks of the ith iteration, i.e, selecting least weight edges from the set $E'(v,c)$, and possible removal of $E'(v,c)$ from E' require a scan of the list of edges E', i.e., $\frac{|E'|}{B}$ I/O operations.

Thus it can be concluded that ith iteration of the first phase of our algorithm can be performed by running integer-sort on an input of size $|E'|$ followed by scanning the set of edges E' a constant number of times.

In order to execute the second phase of the algorithm in external memory, we perform $Sort(E', f_{C_{\lfloor \frac{k}{2} \rfloor -1}}, f_{C_{\lfloor \frac{k}{2} \rfloor}})$ or $Sort(E', f_{C_{\lfloor \frac{k}{2} \rfloor}}, f_{C_{\lfloor \frac{k}{2} \rfloor}})$ depending on whether k is even or odd. This will arrange the set of inter-cluster edges $E(c,c')$ between two neighboring clusters c and c' in contiguous positions, so that the least weight edge from this set can be computed in $\frac{|E'(c,c')|}{B}$ I/O operations.

Theorem 5.1 *Given a weighted graph $G(V,E)$, and an integer k, there exists an external-memory algorithm for computing $(2k-1)$-spanner that has same running time as that of external-memory sorting on an input of size $|E|$.*

5.2 Implementation in Parallel environment

We shall first give a parallel algorithm for a priority CRCW model, which is a very strong PRAM model, and later for the EREW model which is a weaker and more practical PRAM model. The algorithm for CRCW PRAM model achieves $O(k)$ time and optimal work (i.e., $O(km)$). In this PRAM model, each processor can be assigned some priority level. The characteristic feature of CRCW model is that multiple processors can write to (or read from) a single memory location. In case of writing, the processor with highest priority succeeds.

For an edge between $u, v \in V$, there are two processors allocated for its two instances : one for the edge $\mathbf{e}(u, v)$ which is associated with the vertex u, and another for the edge $\mathbf{e}(v, u)$ which is associated with the vertex v. Let A_i be the array associated with the function $f_{\mathcal{C}_i}$ of a clustering $\mathcal{C}_i$. The relative priority between two processors associated with two different edges is defined as follows : the processor P_e associated with the edge e has higher priority than a processor $P_{e'}$ associated with an edge e', iff the weight of edge e is less than that of the edge e'. There is a processor associated with each vertex also.

It is easy to observe that the sampling of clusters can be performed in a single step since each cluster selects itself in the sample independent of the others. Also computation of the nearest sampled neighboring cluster can be performed concurrently for each vertex as follows. For a vertex u, let $e_1, e_2, \cdots e_j$ be the edges incident on u from the clusters sampled in ith iteration. Each of the processors $P_{e_l}, 1 \leq l \leq j$ attempt to write the label of their sampled cluster into location $A_i[u]$, and the processor associated with the least weight edge succeeds.

In a similar way, the least weight edge between a vertex and an unsampled cluster can be computed in a single CRCW step.

Hence all the operations of an iteration of the first phase of the algorithm for computing $(2k-1)$-spanner can be performed in a constant number of parallel steps by a set of $m+n$ processors in CRCW PRAM model. There being $\lfloor \frac{k}{2} \rfloor$ such iterations, it can be seen a $(2k-1)$-spanner can be computed in CRCW PRAM model. in $O(k)$ steps and performing $O(mk)$ work,

Theorem 5.2 *Given a weighted graph with m edges and an integer k, a $(2k-1)$-spanner can be computed in CRCW PRAM model in $O(k)$ time using $O(m)$ memory space and performing $O(km)$ work.*

Consider the following basic Lemma for simulating a CRCW PRAM algorithm in EREW PRAM.

Lemma 5.1 *[12] Single step of p processors in priority CRCW PRAM model with $O(m)$ memory can be executed in $O(\log p)$ steps by p processor in EREW PRAM model with $O(m+p)$ memory.*

The following Theorem follows from Lemma 5.1 and Theorem 5.3.

Theorem 5.3 *Given a weighted graph with m edges and an integer k, a $(2k-1)$-spanner can be computed in EREW PRAM model in $O(k \log n)$ time using $O(m)$ memory space and performing $O(km \log n)$ work.*

References

[1] I. Althófer, 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. Bollobás. In *Extremal Graph Theory*, page 164. Academic Press, 1978.

[5] J. Bondy and M. Simonovits. Cycle of even length in graphs. *Journal of Combinatorial Theory, Series B*, 16:97–105, 1974.

[6] E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM J. Comput.*, 28:210–236, 1998.

[7] P. Erdős. Extremal problems in graph theory. In *Theory of Graphs and its Applications(Proc. Sympos. Smolenice,1963)*, pages 29–36. Publ. House Czechoslovak Acad. Sci., Prague,1964, 1963.

[8] S. Halperin and U. Zwick. Unpublished result. 1996.

[9] D. Peleg and A. A. Schaffer. Graph spanners. *Journal of Graph Theory*, 13:99–116, 1989.

[10] D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. *SIAM Journal of Computing*, 18:740–747, 1989.

[11] D. Peleg and E. Upfal. A trade-off between space amd efficiency for routing tables. *Journal of Assoc. Comp. Mach.*, 36(3):510–530, 1989.

[12] J. H. Reif. In *Synthesis of Parallel Algorithms*, page 852. Morgan Kaufmann Publishers, 1993.

[13] J. D. Salowe. Construction of multidimensional spanner graphs, with application to minimum spanning trees. In *ACM Symposium on Computational Geometry*, pages 256–261, 1991.

[14] M. Thorup and U. Zwick. Approximate distance oracle. In *Proceedings of 33rd ACM Symposium on Theory of Computing (STOC)*, pages 183–192, 2001.

[15] U. Zwick. Personal communication.