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 $G(V, E_S)$, $E_S \subset E$, such that the distance between any pair of vertices in the sub-graph 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 compute spanners with size matching the conjectured lower bound, and the previously 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. Note that *linear time* is a severe limitation since it implies that only a negligible fraction of the distances can be exploited in the construction of the spanner. Nonetheless, all n^2 distances must be faithfully approximated.

An important feature of our algorithm is its local approach. Unlike all the previous algorithms that 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 near optimal.

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. More precisely, 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 the underlying graph structures. For example, in order to maintain compact routing tables [13], many

*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.

existing routing schemes use the edges of a sparse spanner for routing messages. Awerbuch [3] and Peleg and Ullman [12] show that the quality of a spanner (in terms of stretch factor and the number of spanner edges) is very closely related to the time and communication complexity of any synchronizer for the network based on this spanner.

An efficient algorithm for computing a sparse spanner may prove to be useful in efficient computation of all-pairs approximate shortest paths also, since the running time of most of the shortest path algorithms is proportional to the number of edges in the graph. Running such an algorithm on a sparse spanner would achieve better computation time at the expense of approximation factor. However, for such an application, it is essential that the task of computing a sparse spanner takes much less time than that of the algorithm for computing all-pairs shortest paths.

Spanners are also used in computational biology [4] in the process of reconstructing phylogeny trees from matrices whose entries represent genetic distances among contemporary living species. For other applications, please refer to the papers [2, 3, 11, 13].

1.1 Lower bound on the size of t-spanner

All the applications of spanners require a t-spanner of smallest possible size. Therefore, from a graph theoretic perspective, the following question arises : How sparse can a t-spanner be? In this regard, a lot of work [11, 15] has been done to establish a lower bound on the size of spanner in terms of its stretch factor. These results use the following simple relationship between the stretch of a spanner and the girth (length of the smallest cycle) of a 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 length at most $2k$. (Alon et al. [1] show that even $\frac{1}{2}n^{1+1/k}$ edges are in fact enough.) It has been conjectured by Erdős [9], Bondy and Simonovits [7], and Bollobás [6] 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. This establishes a lower bound of $\Omega(n^{1+1/k})$ on the worst case size of a spanner with stretch $2k$ (or $2k-1$).

1.2 Existing algorithms for computing t-spanner

Althöfer et al. [2] gave the first algorithm for computing a t-spanner for weighted graphs. Their algorithm is similar to Kruskal's algorithm for computing a minimum spanning tree. The algorithm builds a spanner gradually as follows : Initially the edges E_S of the spanner is $\emptyset$. The edges of the graph are processed in the increasing order of their weights. The decision as to whether an edge, say $e(u,v)$ has to be added (or not) to E_S is made as follows:

If the distance between u and v in the graph induced by the current spanner edges E_S (the spanner edges selected prior to $e(u,v)$) is more than $t \cdot weight(e(u,v))$, then we select and add the edge to the spanner, otherwise we discard the edge.

It is easy to observe that at the end of this process, the graph $G(V, E_S)$ will be a t-spanner. Moreover, the girth of the graph $G(V, E_S)$ is at-least $t+1$. Since a graph with more than $n^{1+1/k}$ edges must have a cycle of at-most $2k$ edges (see [1]). Hence the algorithm of Althöfer et al. [2] computes a $(2k-1)$-spanner of size $O(n^{1+1/k})$, which is indeed optimal based on the lower bound mentioned earlier. However,

the best known implementation of the algorithm of Althöfer et al. [2] for $(2k-1)$-spanner has a running time of $O(mn^{1+1/k})$. Cohen [8] presented a randomized algorithm with $O(kmn^{1/k})$ expected running time for computing a spanner of $O(kn^{1+1/k})$ size and slightly larger stretch $(2k+\epsilon)$. Thorup and Zwick [15], improved the result of Cohen [8], and presented 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 [2], or computing shortest path trees from a set of $(n^{1/k})$ vertices [15]. Since there is a bound of $\tilde{O}(m)$ on the best known algorithm for computing a shortest path tree, the earlier algorithms for spanners are unlikely to achieve a running time of $O(m)$.

Better bounds are known for spanners of special classes of graphs, e.g., chordal graphs, unweighted graphs, Euclidean graphs in d-dimensions. For chordal graphs, Peleg and Schaffer [11] showed a lower bound of $O(n^{3/2})$ on the size of 2-spanner and gave an algorithm for computing a 2-spanner with size matching this lower bound. They also presented an algorithm for computing a 3-spanner of $O(n \log n)$ size for chordal graphs. For unweighted graphs, Halperin and Zwick [10, 16] gave an $O(m)$ time algorithm to compute a $(2k-1)$-spanner of $O(n^{1+1/k})$ size. Salowe [14] presented an algorithm for computing a $(1+\epsilon)$-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 extend to general weighted undirected graphs.

1.3 Our contribution

We present a randomized algorithm that takes expected linear time for computing a $(2k-1)$-spanner of optimal size for a given weighted undirected graph. More specifically we show that (c.f. Theorem 4.3)

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

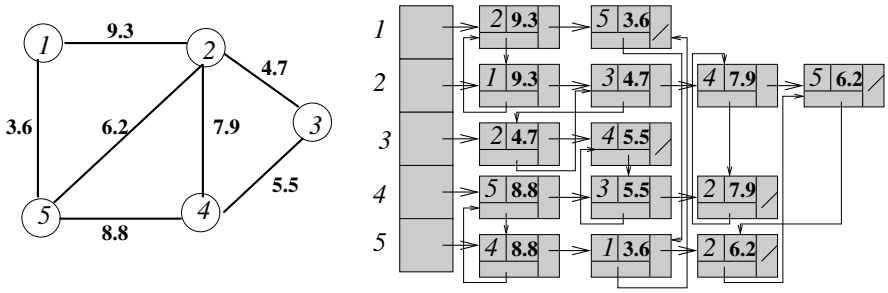
Unlike the previous algorithms that require global information (computing full shortest path tree from few vertices or determining the pair-wise distances), our algorithm requires only local information, i.e., in the neighborhood of each vertex or a group of vertices.

In addition to achieving a linear time sequential algorithm for computing a $(2k-1)$-spanner, the local approach of our algorithm leads to simple near optimal algorithms for computing a $(2k-1)$-spanner in parallel computation environment and in external memory as well. For priority CRCW PRAM model, we present a parallel algorithm that computes a $(2k-1)$-spanner with optimal speed-up in $O(k\tau)$ steps for any $\tau \geq \log^* n$. 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.

1.4 Organization of the paper

We have organized this 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 expose some of the key ideas (clustering of vertices) that we formalize and extend in section 3. We present our linear time sequential algorithm for computing $(2k-1)$-spanner in section 4. We outline the parallel and external-memory algorithms for computing sparse spanners in section 5.

Throughout the paper, unless stated otherwise, we assume that the undirected graph has the augmented adjacency-list representation, wherein for each edge $e(u, v)$, the two nodes associated with the edge (in the adjacency lists of each of u and v) have addresses of each other. See Figure 1 given below.



(a) (b)

Figure 1: (a) an undirected weighted graph (b) an augmented adjacency list representation

This representation will be helpful in the following way. The graph is undirected and therefore, an edge, say $e(u, v)$ appears twice : once in the adjacency list of each of u and v. While processing vertex u, if we decide to delete an edge $e(u, v)$ from the graph, we have to delete the edge from the adjacency list of vertex v too. The above representation makes it possible to perform this operation in constant time.

If the initial graph has simple adjacency list representation, we can get its augmented adjacency list representation in $O(m)$ processing time. Without loss of generality, it is also assumed that all edge weights are distinct.

2 Computing a 3-spanner

In order to compute a 3-spanner of a given weighted graph $G(V, E)$, the objective is to select at most $O(n^{3/2})$ edges to be included in the spanner out of (potentially $\theta(n^2)$) edges E of the graph, and still ensure that the distance between any pair of vertices in the spanner increases to no more than three times their actual distance. To meet the size constraint of 3-spanner a vertex, on an average, should contribute $\sqrt{n}$ edges to the spanner. So the vertices with degree $O(\sqrt{n})$ are easy to handle since we can select all their edges as spanner-edges. The vertices with higher degree pose the following problem : *which $O(\sqrt{n})$ edges should be chosen out of potentially $\theta(n)$ edges incident on a (high degree) vertex ?* We employs a novel clustering scheme for such vertices. Our algorithm computes a 3-spanner in the following two phases.

1. *Forming the clusters* :
   We choose a sample $\mathcal{R} \subset V$ by picking each vertex independently with probability $\frac{1}{\sqrt{n}}$. 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 [1] to v. We add the edge $e(v, \mathcal{N}(v, \mathcal{R}))$ to the spanner along with every other edge that

[1] Ties can be broken arbitrarily. However, it helps conceptually to assume that all weights are distinct

is incident on v with weight less than that of $e(v, \mathcal{N}(v, \mathcal{R}))$. The vertex v is added to the cluster centered at $\mathcal{N}(v, \mathcal{R})$.
We discard all edges $e(u, v)$ (i.e. they are not in E_S) where u and v are not sampled and they belong to the same cluster.

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'. A few observations worth mentioning at this point are the following

: Each vertex of set V' is either a sampled vertex or adjacent to some sampled vertex, and the first phase has partitioned V' into disjoint clusters each centered around some sampled vertex. Consequently each edge from the set E' is an inter-cluster edge and each vertex $v \in V'$ is adjacent to at-most $|\mathcal{R}|$ clusters even though there could be potentially $\theta(n)$ vertices adjacent to it in the graph $G(V', E')$.

The graph $G(V', E')$ is passed onto the second phase.

2. *Joining vertices with their neighboring clusters*:
We process each vertex u of graph $G(V', E')$ as follows. We scan the adjacency list of u. Let $E'(u, c)$ be the edges from the set E' incident on u from a cluster c. We select and add the least-weight edge from $E'(u, c)$ 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. Let us first bound the sizes of E_S^1 and E_S^2 which is relatively easier to prove than the bounds on stretch.

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 edges E_S^2 added to the spanner in the second phase 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 $O(n^{3/2})$.

We will now show that $E_S^1 \cup E_S^2$ has the required properties of a 3-spanner. From the description of the first phase of the algorithm, the following Lemma holds.

Lemma 2.1 *The weight of each edge $e(u, v) \in E \backslash E_S^1$ 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)*

The closeness of vertices of a cluster towards its center relative to the external vertices (as mentioned in Lemma 2.1) is used in the following lemma to bound the stretch of the spanner by 3.

Lemma 2.2 *For an edge $e(u, v) \in E$ that is not present in the spanner $G(V, (E_S^1 \cup E_S^2)$, 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 $e(u, v)$.

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 u (as well as v) belong to some cluster. There are two cases now.
Case 1 : *(u and v belong to same cluster)*
Let u and v belong to the cluster centered at $x \in \mathcal{R}$. It follows from Lemma 2.1 that there is a 2-edge path

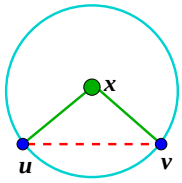


Figure 2: vertex u belongs to the same cluster as the vertex v

$u - x - v$ in the spanner with edge-weight no more than the weight of the edge $e(u, v)$. 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 : *(u and v belong to different clusters)*
Suppose the edge $e(u, v)$ was removed while processing the vertex u, and let v belong to the cluster centered at $x \in \mathcal{R}$ (see Figure 3).

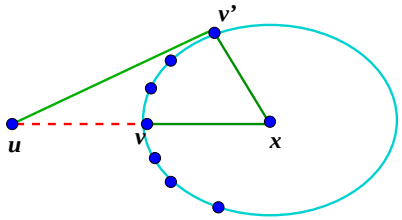


Figure 3: vertex u does not belong to the cluster containing vertex v

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

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

□

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 Π_{uv} between the two in the graph $G(V, E)$. For each edge e on 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 vertices in the graph $G(V, E)$.

Lemma 2.3 *Both phases of the algorithm for computing a 3-spanner can be executed 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 first phase is to select (and add to the spanner), for each vertex v, all the edges incident with weight less than that of the edge $e(v, \mathcal{N}(v, \mathcal{R}))$. This can be performed by traversing adjacency list of each vertex.

In the second phase of the algorithm, for each vertex v and a cluster neighboring to v, we have to select the least weight edge between the two. 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 $e(v, w)$ 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 $N[w]$). If $A[x]$ was pointing to *null* previously, we make $A[x]$ point to the edge $e(v, w)$. Otherwise let $A[x]$ was pointing to an edge, say $e(v, y)$ previously. We shall make $A[x]$ point to the lighter (having less weight) of the two edges $e(v, y)$ and $e(v, w)$. It is easy to verify that once all the edges incident on v have been processed, the adjacency list of v consists of only the least weight edges between v and its neighboring clusters. Now we perform another scan of this adjacency list, and for each edge $e(v, w)$ we add the edge pointed by $A[N[w]]$ to the spanner. We also make $A[N[w]]$ 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 its initial state (all its entries pointing to *null*) to be used for another vertex.

Thus with an extra space (arrays A and N) of $O(n)$ size, 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 $(2k-1)$-spanner 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 $e \in E \backslash E_S$ 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 $G(V, E)$ reduces[2] to finding a subset $E_S \subset E$ such that $\mathcal{P}_{2k-1}(e)$ holds for each edge $e \in E \backslash E_S$. 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 is the partitioning of the set of vertices into *suitable clusters*. Recall from the previous section how the clustering of the vertices (by 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 was the smaller number of these clusters compared to the number of vertices

[2]Note that it implies a stronger property than required by a spanner

that helped in getting a bound on the size of the 3-spanner, and it was the closeness of the vertices within a cluster that ensured a bound on the stretch of the spanner.

Our algorithm for computing a $(2k-1)$-spanner employs a *clustering induced by a set of edges*. We now formally define this clustering and 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* **clustering** *of $V' \subset V$ is a partition of V' into clusters. 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**. *Formally, a clustering $\mathcal{C}$ can be represented by a function $f_{\mathcal{C}} : V \to V$ such that $f_{\mathcal{C}}(u)$ is the center of the cluster to which the vertex u belongs. Note that $f_{\mathcal{C}}(u) = f_{\mathcal{C}}(v)$ iff vertices u and v belong to same cluster. Hence the function $f_{\mathcal{C}}$ associated with the clustering $\mathcal{C}$ can be used to determine whether any two vertices belong to the same cluster or not.*

We now define a clustering induced by a set of edges :

Definition 3.2 *Given a graph $G(V, E)$, a set of edges $\mathcal{E} \subset E$ induces a partition of set V into clusters in the following natural way : two vertices belong to a cluster if they are connected by a path $\Pi \subset \mathcal{E}$. (In other words, each connected component is a cluster). We call this clustering as the clustering induced by $\mathcal{E}$. For a cluster c in this clustering, we shall use $\mathcal{E}(c) \subset \mathcal{E}$ to denote the edges defining the connected component associated with the cluster c.*

Definition 3.3 *Consider a clustering $\mathcal{C}$ induced by $\mathcal{E} \subset E$ in a given graph $G(V, E)$. The radius of a cluster $c \in \mathcal{C}$ is the smallest integer r such that the following holds :*

For each edge $e(x, y) \in E \backslash \mathcal{E}$, $x \in c$, there is a path $\Pi \subset \mathcal{E}(c)$ from x to $f_{\mathcal{C}}(x)$ of at most i edges each having weight no more than that of the edge $e(x, y)$.

Definition 3.4 *A clustering $\mathcal{C}$ induced by $\mathcal{E} \subset E$ is a clustering of radius $\leq i$ in the graph $G(V, E)$ if each of its cluster has radius $\leq i$.*

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

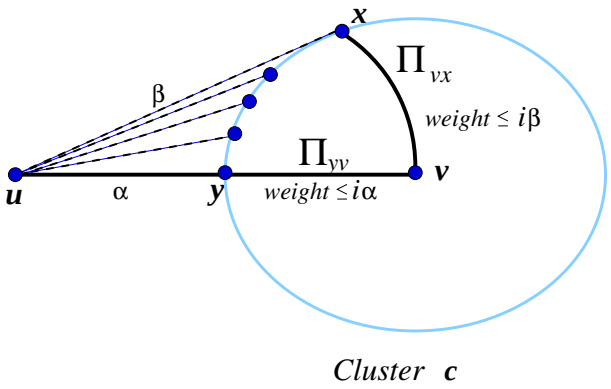


Figure 4: Ensuring that the proposition $\mathcal{P}_{2i+1}$ holds for the set $E'(u,c)$.

Lemma 3.1 *Let $\mathcal{C}$ be a clustering of radius i induced by $\mathcal{E}$ in a given graph $G(V', \mathcal{E} \cup E')$, and let $c \in \mathcal{C}$ be a cluster. If set $\mathcal{E}$ has been included in the spanner, then for any vertex $u \notin c$, picking the least weight edge from the set $E'(u,c)$ in the spanner will ensure that the proposition $\mathcal{P}_{2i+1}(e)$ holds for each edge $e \in E'(u,c)$.*

Proof: Let the edge $e(u,y)$ of weight α be the least-weight edge from the set $E'(u,c)$. Let (u,x) be any other edge of weight $\beta \geq \alpha$ from the set $E'(u,c)$ (see Figure 4). Since the radius of the cluster c is at-most i, therefore, there is a path $\Pi_{vx} \subset \mathcal{E}(c)$ between vertex x and the center v of the cluster c, and its weight is at-most i times β. Using the same argument, we deduce that there is a path $\Pi_{yv} \subset \mathcal{E}(c)$ from vertex y to v with weight at-most $i\alpha$. Thus there is path Π_{ux} from vertex u to vertex x formed by concatenating the edge $e(u,y)$ and the paths Π_{yv}, Π_{vx} in this order; and its weight can be bounded as follows.

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

Therefore, we can conclude that if $\mathcal{E}$ has been included in the spanner, then adding the edge $e(u,y)$ to the spanner makes the proposition $\mathcal{P}_{2i+1}(e)$ true for each edge $e \in E'(u,c)$. □

Along similar lines we can prove the following Lemma.

Lemma 3.2 *Let $\mathcal{C}$ be a clustering induced by $\mathcal{E}$ in a given graph $G(V', \mathcal{E} \cup E')$, and let $c_1, c_2 \in \mathcal{C}$ be two clusters having radius i and j respectively. If set $\mathcal{E}$ has been included in the spanner, then picking the least weight edge from the set $E'(c_1,c_2)$ in the spanner 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 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 $k-1$ iterations. During ith iteration, a set of edges E_S^i are selected to be added to the spanner that makes the proposition $\mathcal{P}_{2i-1}$ true for a possibly large set of edges using Lemma 3.1. All these edges are removed from the graph. In successive iterations, the size of the clustering, i.e., the number of clusters reduces by a factor of $n^{1/k}$ while the radius of clusters increases by just one unit.

At the end of $k-1$ iterations, we obtain a clustering that consists of only $n^{1-\frac{k-1}{k}} = n^{1/k}$ clusters. 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 called *Termination*, each vertex is joined to the neighboring clusters using the least weight edge (as in the case of 3-spanner).

In another variation called *Joining the cluster*, the clusters are joined together by adding the least weight edge between each pair of neighboring clusters after only $\lfloor \frac{k}{2} \rfloor$ iterations. At this stage, there are $n^{1-\frac{1}{k}\lfloor \frac{k}{2} \rfloor}$ clusters, and a subset of spanner-edges added till $\lfloor \frac{k}{2} \rfloor$th iteration ensures that the radius of each cluster is no more than $\lfloor \frac{k}{2} \rfloor$. 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 $k-1$ iterations, and selects E_S^i spanner edges in ith iteration. In the beginning of ith iteration, we have the set E' of edges for which the proposition $\mathcal{P}_{2i-1}$ does not hold yet and V' is the set of their endpoints. Moreover there is a small subset $\mathcal{E}_{i-1}$ of spanner edges that induces a clustering $\mathcal{C}_{i-1}$ in the graph $G(V', \mathcal{E}_{i-1} \cup E')$.

Initially, i.e., in the beginning of the first iteration the sets are $E_S^0, \mathcal{E}_0 = \emptyset, E' = E, V' = V$, and the clustering is $\mathcal{C}_0 = \{\{v\}|v \in V\}$.

The ith iteration performs the following four steps in the fixed order.

1. *Forming a sample of clusters* : A sample $\mathcal{R}_i$ of clusters is chosen by picking each cluster from the clustering $\mathcal{C}_{i-1}$ independently with probability $n^{-\frac{1}{k}}$. The set $\mathcal{E}_i$ is initialized to those edges of set $\mathcal{E}_{i-1}$ that define the clusters of $\mathcal{R}_i$. As a consequence, the clustering $\mathcal{C}_i$ is initialized to $\mathcal{R}_i$.

2. *Finding nearest neighboring cluster for all vertices* : For each vertex $v \in V'$ not belonging to any sampled cluster, compute nearest neighboring cluster from set $\mathcal{R}_i$.

3. *Selecting edges E_S^i for spanner* : To select the spanner edges in ith iteration, process each vertex $v \in V'$ not belonging to any sampled cluster according to the following two cases.

   (a) If v is not adjacent to any sampled cluster, then for each cluster $c \in \mathcal{C}_{i-1}$ adjacent to v, we add the least weight edge from the set $E'(v, c)$ to E_S^i, and remove the edges $E'(v, c)$ from set E'.

   (b) If vertex v is adjacent to one or more sampled clusters, let $c \in \mathcal{R}_i$ be the cluster that is adjacent to v with edge of least weight (say e_v) among all the clusters from the set $\mathcal{R}_i$. We add the edge e_v to the sets E_S^i and $\mathcal{E}_i$ [3], and remove the entire set $E'(v, c)$ from E'. In addition, we do the

[3] This ensures that $\mathcal{E}_i$ is always a subset of spanner.

following. For each cluster $c' \in \mathcal{C}_{i-1}$ adjacent to vertex v with an edge of weight less than that of e_v, we add the least weight edge from the set $E'(v, c')$ to E^i_S, and remove $E'(v, c')$ from E'.

After this 3rd step of ith iteration, note that only those edges remain in the set E' whose end-points either belong to or are adjacent to some sampled cluster in $\mathcal{R}_i$.

Observation 4.1 *It follows from the way we construct $\mathcal{E}_i$ (see step 1 and 3(b) of the algorithm) that the clustering $\mathcal{C}_i$ induced by $\mathcal{E}_i$ is such that a cluster $c \in \mathcal{C}_i$ is the union of a sampled cluster $R \in \mathcal{R}_i$ with the set of all those vertices for which R was the nearest neighboring sampled cluster.*

4. *Removing intra-cluster edges* : All the intra-cluster edges (whose both end-points belong to the same cluster) of the clustering $\mathcal{C}_i$ are eliminated from E'.

Observation 4.2 *During an iteration, a cluster is sampled and passed on to the next iteration with probability $n^{-\frac{1}{k}}$. Therefore, the expected number of the clusters at the end of ith iteration is $n^{1-\frac{i}{k}}$. That is, $|\mathcal{C}_i| = n^{1-\frac{i}{k}}$.*

Observation 4.1 gives a formal description of the clusterings (*partitioning* of vertices for the next stage) defined in the successive iterations of the algorithm. The following invariant is a key to understanding of the way the algorithm works.

Theorem 4.1 *The following assertion holds for each iteration $j \geq 0$.*
The clustering $\mathcal{C}_j$ induced by the set $\mathcal{E}_j$ in the algorithm is a clustering of radius j in $G(V', E' \cup \mathcal{E}_j)$.

Proof: We shall prove the theorem by induction on $j \geq 0$.
Base Case : $j = 0$ **:** In the beginning of the algorithm, $\mathcal{E}_0 = \emptyset$, the clustering is $\mathcal{C}_0 = \{\{v\} | v \in V\}$, and $V' = V, E' = E$. It is easy to observe that each cluster in $\mathcal{C}_0$ is a cluster of radius 0 in the graph $G(V, E)$. Therefore, the assertion $\mathcal{A}(0)$ holds.
Induction Hypothesis : $j < i$ **:** Let the assertion $\mathcal{A}(i-1)$ holds.
Proof of assertion $\mathcal{A}(i)$:
As observed at the end of step *3* of the ith iteration, a cluster from the clustering $\mathcal{C}_i$ is a union $R \cup N_R$ of a cluster $R \in \mathcal{R}_i$ with the set N_R of all those vertices of set V' for whom the cluster R is the nearest neighboring sampled cluster (see Figure 5). The center of the cluster $R \cup N_R$ is same as that of the cluster R (see Definition 3.1). It follows from the induction hypothesis that R is a cluster of radius $i-1$ in the clustering induced by $\mathcal{E}_{i-1}$. That is, for each edge $e(x, y) \in E', x \in R$, there is a path $\Pi \subset \mathcal{E}_{i-1}(R)$ from x to the center o in the cluster R consisting of at-most $i-1$ edges each of weight no more than that of $e(x, y)$. Since the edges of set $\mathcal{E}_{i-1}(R)$ are present in $\mathcal{E}_i$ too (see step 1 of the algorithm), so the radius of cluster R is $i-1$ even in the clustering induced by $\mathcal{E}_i$.

Now consider a vertex $v \in N_R$. During ith iteration, we add to the set $\mathcal{E}_i$ (and to the spanner), the edge e_v of least weight from the set $E'(v, R)$. Let $u \in R$ be the second endpoint of edge e_v. Therefore, there is a path $\Pi_{vo} \in \mathcal{E}_i$ formed by concatenating e_v with $\Pi_{uo} \in \mathcal{E}_i$ that consists of at-most i edges and the weight of each edge on this path is no more than that of e_v (invoke induction hypothesis with $x = u$ and $y = v$). Moreover, as can be noticed from step 3(b) of the algorithm, there is no edge left in the set E' which is incident on v with weight less than that of e_v. Hence weight of each edge on the path Π_{v-o} is infact no more than any edge $e(v, z) \in E'$ at the end of ith iteration. These statements hold true for each $v \in N_R$. Hence $R \cup N_R$ is a cluster of radius i in the graph $G(V', \mathcal{E}_i \cup E')$.

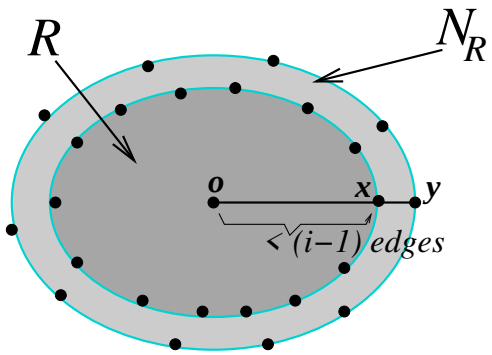


Figure 5: A cluster at the end of ith iteration

Similar arguments can be given for any other cluster in the clustering $\mathcal{C}_i$. Thus the assertion $\mathcal{A}_i$ holds. Hence by the principle of mathematical induction, the assertion $\mathcal{A}_j$ holds for all $j \geq 0$. □
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 $\mathcal{P}_{2k-2}$ holds.*

Proof: Let $e(u, v)$ 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 fourth step of ith iteration.
Case 1 : (*The edge $e(u, v)$ is eliminated from E' during step 3*)
Without loss of generality, let the edge $e(u, v)$ get eliminated while processing the vertex u (the case for v is similar). Note that we surely add the least weight edge between 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 $e(u, v)$.
Case 2: (*The edge $e(u, v)$ is eliminated during step 4*).

In this case both u and v must have been assigned to same cluster, say $c \in \mathcal{R}_i$. It follows from the step 3 of the ith iteration that the edge $e(u, v)$ is at-least as heavy as the edge $\min(E'(u, c))$ that we add to the spanner. Moreover Theorem 4.1 implies that c is a cluster of radius $i - 1$. Therefore, there is a path Π_{u-o} (likewise Π_{v-o}) from u (likewise v) to the center o of the cluster c consisting of at-most i spanner-edges, each of weight no more than that of $e(u, v)$. Thus the path formed by concatenating the paths Π_{u-o}, Π_{o-v} in this order is a path between u and v consisting of at-most $2i$ spanner-edges, each of weight no more than that of $e(u, v)$. In other words $\mathcal{P}_{2i}$ holds for the edge $e(u, v)$.

Since there are $k - 1$ iterations in the first phase, it follows that $\mathcal{P}_{2k-2}$ holds for each edge eliminated from the graph in the first phase. □

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: Let v be a vertex belonging to set V' during ith iteration of the first phase. All the neighbors of the vertex v are grouped into their respective clusters of the clustering $\mathcal{C}_{i-1}$. 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 the set $E'(v, c_j)$ is lighter (has less weight) than the least weight edge from the set $E'(v, c_{j+1})$ 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 $n^{-1/k}$, so 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 $v \in V'$ 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 exceeds $O(n^{1+1/k})$ - the expected number of repetitions is $O(1)$. There are total $k-1$ iterations in the first phase, 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 random 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 steps 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 $k-1$ iterations in the first phase, therefore, the expected running time of the first phase is $O(km)$. □

Remark : For an unweighted graph, each vertex would add only single edge to the spanner in each iteration of the first phase except the iteration in which it gets eliminated; during this iteration the vertex contributes $n^{1/k}$ edges. Hence the expected number of edges added to the spanner in the first phase will be $O(n^{1+1/k})$ if the graph is unweighted.

Phase 2: Termination

At the end of the first phase of the algorithm, let $E_S^1 \cup E_S^2 \ldots E_S^{k-1}$ be the set of edges added to the spanner and V' and E' be the set of vertices and edges respectively left in the graph. For each vertex $v \in V'$ and a neighboring cluster $c \in \mathcal{C}_{k-1}$, we choose the least weight edge among $E'(v, c)$ and add it to the spanner - denote this set of edges by E_S^k. The remaining edges (inter-cluster) are discarded and we output the set $E_S = E_S^1 \cup E_S^2 \ldots E_S^k$ as $2k-1$ spanner.

Since there are at most $n^{1/k}$ clusters at this stage (observation 4.2) E_S^k is $O(n^{1+1/k})$. For each edge deleted in the second phase, we apply the same argument as Case 1 in the proof of Theorem 4.2. Since $i = k$, proposition $\mathcal{P}_{2k-1}$ holds for every edge eliminated.

Execution of this phase is similar to the execution of an iteration of the first phase (with each cluster in $\mathcal{C}_{k-1}$ sampled with probability 1 in step 1), so its running time is $O(m)$. We can now state the main theorem.

Theorem 4.3 *Given a weighted graph $G(V, E)$, and integer $k > 1$, a spanner of $(2k-1)$ stretch and $O(kn^{1+1/k})$ size can be computed in expected $O(km)$ time (for unweighted graph, the size of spanner is $O(n^{1+1/k})$).*

4.3 An alternative to second phase : Joining the clusters

There can be a slight variation in the algorithm described in previous subsection that can save a factor of 2 in the number of edges in $(2k-1)$-spanner. This does not give us any asymptotic improvement in the size. However, for unweighted graphs, this variation would ensure a stretch which is strictly less than $2k-1$ for all paths of length more than one. The variation in the algorithm is the following. We don't execute all $k-1$ iterations of the first phase. Instead, we stop after $\lfloor \frac{k}{2} \rfloor$ iterations, and then as an alternative to the second phase, we join each pair of neighboring clusters with the least weight edge between them. The new algorithm is described below.

1. Do $\lfloor \frac{k}{2} \rfloor$ iterations in the first phase.

   The set $\mathcal{E}_{\lfloor \frac{k}{2} \rfloor}$ of edges partitions the vertices V' (corresponding to the end-points of E') into 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 4.2). Moreover, the Theorem 4.1 implies that $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ 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 following phase of the algorithm.

2. *Joining the clusters* :

   - 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 to the spanner. To execute this, first we merge the adjacency lists of all the vertices belonging to same cluster in the clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$. We process the merged list associated with a cluster c as follows. For each cluster $c' \in \mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ incident on c, we select and add the least weight edge from set $E(c, c')$ to the spanner.
   - If k is even, then for each pair of neighboring clusters $c \in \mathcal{C}_{\lfloor \frac{k}{2} \rfloor}, c' \in \mathcal{C}_{\lfloor \frac{k}{2}-1 \rfloor}$, we add the least-weight edge between the two clusters to the spanner. To execute this, first we merge the adjacency lists of all the vertices belonging to same cluster in the clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$. For each cluster $c' \in \mathcal{C}_{\lfloor \frac{k}{2}-1 \rfloor}$ incident on c, we select and add the least weight edge from set $E(c, c')$ to the spanner.

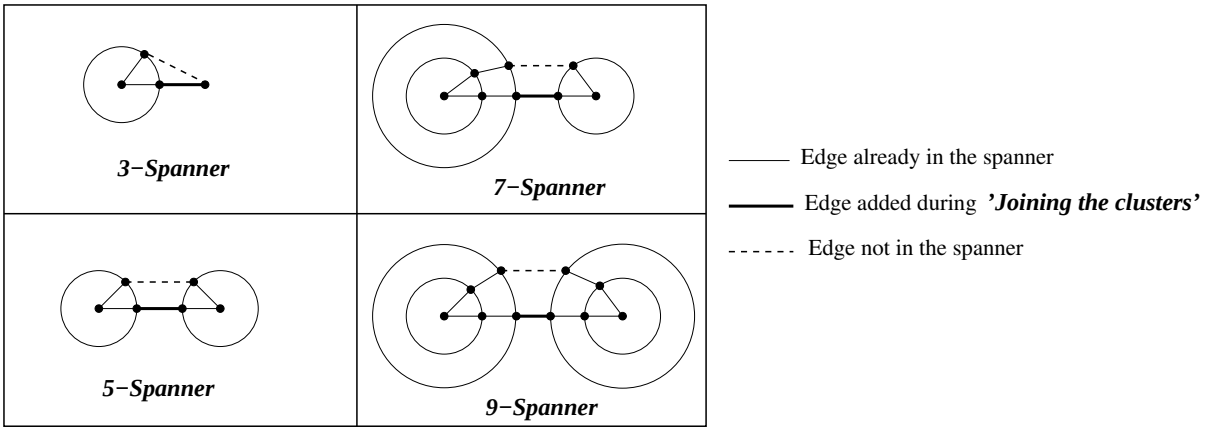


Figure 6: *'Joining the clusters'* phase of the new algorithm

Figure 6 shows how the clusters are joined in *'Joining the clusters'* phase of our new algorithm for building the spanners of stretch 3,5,7 and 9 (i.e., $k = 2, 3, 4, 5$).

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 *'Joining the clusters'* phase. So the new algorithm indeed computes a $(2k-1)$-spanner. Also note that for both odd and even cases of k as described above, the processing of a cluster $c \in \mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ is similar to the way we process a vertex in the second phase of the algorithm for 3-spanner (see Section 2). Hence the new algorithm still has a running time of $O(km)$.

Lemma 4.2 *The total number of edges added to the spanner during the 'Joining the clusters' 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 4.2.
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 using Observation 4.2, the total number of edges added to the spanner in the second phase is $n^{1+1/k}$. □

Spanners with improved stretch for unweighted graphs :

For unweighted graphs, there is a concept of (α, β)-spanner as defined below.
Definition 4.3: An (α, β)-spanner of a graph $G(V, E)$ is a *subgraph* $G(V, E_S)$ such that for all vertices $u, v \in V$, the distance $\delta^*(u, v)$ between them in the spanner is related to their actual distance $\delta(u, v)$ as follows:

$$\delta^*(u, v) \ \leq \ \alpha \cdot \delta(u, v) + \beta$$

It can be seen that $(2k-1)$-spanner defined earlier is indeed a $(2k-1, 0)$-spanner. In an (α, β)-spanner a single edge may be stretched by as much as $\alpha + \beta$. However the average stretch in a long path is as low as α. It is desirable to have smaller multiplicative stretch α at the cost of increased additive stretch β.

Now we shall show that $(2k-1)$-spanner of an unweighted graph computed by our new algorithm described above is indeed a $(\frac{3}{2}k, k-1)$-spanner. In other words, long paths are stretched by a factor of $\approx \frac{3}{2}k$ only.

We shall use the following lemma that follows immediately from *'Joining the clusters'* phase of the new algorithm.

Lemma 4.4 *Let $G(V, E)$ be a given unweighted graphs and $G(V, E_S)$ be a $(2k-1)$-spanner as computed by the new algorithm. If $e(x, y)$ is an edge, and both x and y belong to clusters $c, c' \in \mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$, then there is a path of length $\leq k$ in the spanner between the center of the clusters c and c'.*

We introduce a notation at this point. For a vertex $x \in V$, $c(x)$ is the vertex x itself if x does not appear in the clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$, otherwise $c(x)$ is the center of the cluster containing x in this clustering. In the latter case, there is a path from x to $c(x)$ of length no more than $\lfloor \frac{k}{2} \rfloor$.

Lemma 4.5 *Let $e(u,v)$ be an edge in a given unweighted graph $G(V,E)$, and $G(V,E_S)$ be a $(2k-1)$-spanner computed by the new algorithm. There is a path $\Pi_{uv} \subset E_S$ between u and v of length at-most $k-2+\lfloor \frac{k}{2} \rfloor$.*

Proof: If neither u nor v belong to the clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ and so $c(u)=u, c(v)=v$, then Lemma 3.1 implies that there is a path of length less than k between $c(u)$ and $c(v)$ in the spanner. If u as well as v belong to some (same or different) clusters in $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$, then Lemma 4.3 implies that there is a path of length at-most k in the spanner between the centers of the two clusters associated with u and v. In other words, $c(u)$ and $c(v)$ are separated by distance at-most k in the spanner.

If u belongs to the clustering $\mathcal{C}_{\lfloor \frac{k}{2} \rfloor}$ while v does not belong (or vice versa), then there is a path from $c(u)$ (same as vertex u) to $c(v)$ of length $k-2+\lfloor \frac{k}{2} \rfloor$ in the spanner : *moving from u to v requires at-most $k-2$ steps using Lemma 3.1 and moving from v to $c(v)$ requires at-most $\lfloor \frac{k}{2} \rfloor$ steps.* □

Let $u=v_0, v_1, \cdots, v_l=v$ be the shortest path Π_{uv} between u and v in the original graph. Consider the sequence $\langle c(v_0), c(v_1), \cdots, c(v_q) \rangle$. It follows from Lemma 4.4 that there is a path in the $(2k-1)$-spanner from $c(u)$ to $c(v)$ of length $(k-2+\lfloor \frac{k}{2} \rfloor)q \leq \frac{3}{2}kq$. Moreover, moving from u to $c(u)$ (likewise moving from $c(v)$ to v) requires at-most $\lfloor \frac{k}{2} \rfloor$ steps in the spanner. Hence, there is a path from u to v in the spanner of length at-most $\frac{3}{2}kq+k-1$. In other words, the spanner is a $(\frac{3}{2}k, k-1)$-spanner.

Theorem 4.4 *An undirected unweighted graph $G(V,E)$ can be processed in linear time to compute a $(\frac{3}{2}k, k-1)$-spanner of size $O(n^{1+1/k})$.*

5 Efficient implementation of the algorithm in various programming environments

The unique aspect of our algorithm is its local approach. At any step, processing of a vertex (or a cluster) merely involves processing of the edges in its local neighborhood. It is due to this local approach that it is quite easy to design near optimal version of this algorithm in external-memory and parallel environment.

Our sequential algorithm for computing $(2k-1)$-spanner performs k iterations. We shall show the following. In external-memory, each such iteration can be executed using a constant number of runs of integer sorting, and in CRCW PRAM model of parallel computation, it requires $\tau \geq \log^* n$ steps in parallel using n/τ processors for each iteration.

Recall that since the graph is undirected, an edge between a vertex u and a vertex v appears twice in the adjacency list representation : once in the adjacency list of u, and once in the adjacency lists of v. In each algorithm of this section, we shall treat the two instances of the edge $u-v$, like two different edges : as edge $e(u,v)$ while processing the vertex u, and as edge $e(v,u)$ while processing the vertex 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, an iteration of our algorithm can be performed in I/O optimal way.

Given two clustering $\mathcal{C}, \mathcal{C}'$ on a set of vertices V', we define a total order $\preceq_{(\mathcal{C},\mathcal{C}')}$ on the set of vertices V' and associated edges E' as follows.

- vertex u precedes vertex v if
$$f_{\mathcal{C}}(u) \leq f_{\mathcal{C}}(v) \quad \text{or} \quad f_{\mathcal{C}}(u) = f_{\mathcal{C}}(v) \text{ and } f_{\mathcal{C}'}(u) < f_{\mathcal{C}'}(v)$$
- edge $e(u,v)$ precedes edge $e(x,y)$ if
$$f_{\mathcal{C}}(u) < f_{\mathcal{C}}(x) \quad \text{or} \quad f_{\mathcal{C}}(u) = f_{\mathcal{C}}(x) \text{ and } f_{\mathcal{C}'}(v) < f_{\mathcal{C}'}(y)$$

Lemma 5.1 *If the list of edges E' are arranged according to the total order $\preceq_{(\mathcal{C},\mathcal{C}')}$, for any two clusters $c \in \mathcal{C}, c \in \mathcal{C}'$,*
(i) The set of edges $\{e(u,v)|u \in c\}$, i.e. the edges emanating from the cluster c appear as a sub-list, say L_c.
(ii)The set of edges $E'(c,c')$ appear as a sub-list within the sub-list L_c.

Data-structure :

- The vertices V' are kept in a list. For each vertex $u \in V'$, we store the following additional variables.
$f_{\mathcal{C}}(u)$: as the center of the cluster containing u.
sampled(u) : a boolean variable which is true during an iteration if u belongs to sampled cluster.
- The data-structure for each edge $e(u,v)$ also has two additional variables. *l-center* storing $f_{\mathcal{C}}(u)$ and *r-center* storing $f_{\mathcal{C}}(v)$.

External-memory algorithm :
The four basic tasks of ith iteration for computing $(2k-1)$-spanner can be performed in external memory as follows.

1. *Forming a sample of cluster*

   We arrange V' and E' according to the total order $\preceq_{(\mathcal{C}_{i-1},\mathcal{C}_0)}$. Consequently, the vertices belonging to same cluster in $\mathcal{C}_{i-1}$ appear together in V'. Scanning the two lists V' and E' simultaneously, we do the following. We pick each cluster for the sample independently with probability $n^{-1/k}$, setting *sampled* variable of the vertices accordingly, and compute the list E'_R of edges incident on vertices of unsampled clusters from sampled clusters.

2. *Finding nearest neighboring cluster for vertices*

   We arrange list E'_R according to the total order $\preceq_{(\mathcal{C}_0,\mathcal{C}_0)}$. As a result, the edges incident on a vertex from all neighboring sampled clusters appear contiguous. We perform a linear scan on this list and among all edges incident on a vertex, we keep only the least weight edge and eliminate all the remaining edges from E'_R.

3. *Selecting edges of E^i_S for spanner*

   - Add E'_R to the spanner.
   - We arrange the edges E' according to total order $\preceq_{(\mathcal{C}_0,\mathcal{C}_{i-1})}$ so that all the edges incident on a vertex from same cluster i clustering $\mathcal{C}_{i-1}$ appear contiguous. Selecting the remaining edges of set E^i_S (and their deletion from set E'), merely requires a simultaneous scan of the lists V', E'_R, E'.

Now, to define the clustering $\mathcal{C}_i$, we perform a simultaneous scan on V' and E'_R after arranging them according to $\preceq_{(\mathcal{C}_0,\mathcal{C}_0)}$. We keep only those vertices $u \in V'$ which have either $sampled(u) = true$ or some edge, say $e(u,v) \in E'_R$ (each deleted vertex is neither a member of some sampled cluster nor adjacent to any sampled cluster). For an edge $e(u,v) \in E'_R$, we need to set $f_{\mathcal{C}}(u) = f_{\mathcal{C}}(v)$ so as to indicate that u is assigned to the cluster containing v in $\mathcal{C}_i$. This can be done by another simultaneous scan of E'_R and V'. The vertices in the list V' along with their variable $f_{\mathcal{C}_i}$ constitute the clustering $\mathcal{C}_i$ for the $(i+1)$th iteration.

For each edge $e(u,v)$ left in the set E', we also need to set its variables *l-center* and *r-center* to $f_{\mathcal{C}}(u)$ and $f_{\mathcal{C}}(v)$ respectively. A simultaneous scan of V' and E' arranged according to $\preceq_{\mathcal{C}_0,\mathcal{C}_0}$ will set the variable *l-center* of each edge. Now rearranging E' so that the two instances of an edge (for example $e(u,v)$ and $e(v,u)$) appear together, we can set the variable *r-center* of each edge by another scan of list E'.

4. *Deleting Intra-cluster edges*

   We scan the list of edges E' and delete every edge if its *l-center* is same as its *r-center*.

Remark While processing a vertex, say u, if we delete an edge $e(u,v)$ from the graph, then for consistency we need to delete the edge $e(v,u)$ also from the adjacency list of v. In case of RAM model, we could perform this task in constant time by using augmented adjacency list where the nodes associated with the edges $e(u,v)$ and $e(v,u)$ had pointers to each others. But this would be a costly operation in external memory since a pointer could lead to a block not present in internal memory. This problem can be overcome as follows. While processing the adjacency list of vertex, say u, if we decide to delete an edge say $e(u,v)$, we do not delete the edge right away. Instead we mark each such edge that has to be deleted. After each step in the iteration, we arrange (sort) the list E' so that the edges with same endpoints appear together in the list. Now, we perform a scan on this list and delete each pair of edges $e(u,v)$ and $e(v,u)$ if any of them is marked for deletion.

Lemma 5.2 *Each iteration of the algorithm for computing $(2k-1)$-spanner can be executed in external-memory in $O(\frac{|E'|}{B}\log_\mu \frac{|E'|}{B})$ I/O-operations.*

Proof: It follows from the description of the algorithm given above that ith iteration arranges V' and E' according to the total orders $\preceq_{(\mathcal{C}_0,\mathcal{C}_{i-1})}$, $\preceq_{(\mathcal{C}_0,\mathcal{C}_0)}$, $\preceq_{(\mathcal{C}_{i-1},\mathcal{C}_0)}$ followed by a scan that is performed in I/O-optimal way. Now it is easy to observe that arranging E' or V' according to any of these total orders requires integer-sort (infact radix-sort) on the labels of the end-points (and/or *l-center* and *r-center*) 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 for the running time as that of generic sorting problem, which is $O(\frac{|E'|}{B}\log_\mu \frac{|E'|}{B})$. Thus we conclude that each iteration of the first phase of our algorithm can be performed in $O(\frac{|E'|}{B}\log_\mu \frac{|E'|}{B})$ I/O-operations. □

5.2 Implementation in parallel environment

The model of computation used by our parallel algorithm is priority CRCW PRAM. This model supports concurrent read as well as write into any memory location by multiple processors. Incase of simultaneous

write operations by two or more processors into a memory location, the processor with highest priority succeeds.

In order design an efficient parallel way to execute each iteration of our algorithm, first we outline a few simple observations. Each of the four tasks of an iteration can be performed in $O(1)$ parallel time easily except the task of selecting the spanner edges E_s^i. This task requires an efficient concurrent way for the following sub-task : *Given a vertex u (with its neighbors belonging to disjoint clusters), select least weight edge per cluster adjacent to it in $O(1)$ time*. If the processors associated with the edges incident on u from same cluster are assigned a common location to write and priority assigned to a processor is inverse of the weight of its edge (so that the processor corresponding to the least weight edge succeeds), then this sub-task can also be performed in $O(1)$ time. Hence the only problem left is : How to allocate a common location for processors whose associated edges emerge from same cluster. There is a trivial solution for this problem if we can allocate $O(n)$ auxiliary space per vertex. But this quadratic space requirement for parallel algorithm is not acceptable. We shall avoid it using a perfect hash function.

Definition 5.3: A perfect hash function for a set X of l integers is an injective function $h : X \rightarrow \{1, \cdots, s\}$, where $s = O(l)$, that can be stored in $O(l)$ space and evaluated in constant time by a single processor.

Bast and Hagerup [5] showed that a perfect hash function for a given set of l integers can be constructed optimally in $O(\tau)$ time using $O(n/\tau)$ processors for any $\tau \geq \log^* n$.

Resources (space and processors) : To each vertex $u \in V$, we allocate

- an auxiliary array A_u of size equal to its neighorhood n_u.
- Two boolean variables :
  sampled(u) : true during an iteration if u belongs to a sampled cluster.
  Is_center(u) : true if u is a cluster center.
  These boolean variables are set to false before the beginning of each iteration.
- *center*(u) : the center of the cluster containing u.
- *lwe*(u) : weight of the least weight edge incident on u from any sampled cluster.
- A set P_u of n_u/τ processors per vertex u for processing the edges incident onto it and one additional processor for processing the vertex u. (The edges incident onto u are evenly distributed among P_u - at-most τ edges per processor)

We shall denote l_p as the set of edges allocated to processor p.

As noticed in the sequential algorithm, an edge is processed once in each iteration until we decide to add or discard that edge from being a spanner-edge (at this stage we have ensured that $\mathcal{P}_{2k-1}$ holds for that edge). From that point onwards, the edge is not considered at all. For this reason, in the parallel algorithm we associate a variable *live(e)* with an edge e which is true as long as we need to process that edge. We also associate a variable *live*(u) with each vertex $u \in V$ in a similar way.

Parallel Algorithm : The four basic tasks of ith iteration for computing $(2k-1)$-spanner can be performed in CRCW PRAM as follows.

1. *Forming a sample of clusters*

   Each cluster center in the clustering $\mathcal{C}_{i-1}$ chooses to remain a cluster center for the next iteration with probability $n^{-1/k}$. Each *live* vertex u of a cluster $c \in \mathcal{C}_i$ determines whether it belongs to a sampled

cluster by performing a concurrent read into the *Is_center* variable of its cluster center, and copying the boolean content into its variable *sampled*(u).

2. *Finding nearest sampled neighboring cluster for vertices*

   For each *live* vertex $u \in V$ with *sampled*(u)=*false*, the task of finding the sampled neighboring cluster nearest to u is accomplished by P_u processors as follows. Each processor $p \in P_u$ processes the list of its live edges l_p sequentially, and for an edge $e(u,v) \in l_p$ with *sampled*(v) = *true*, the processor p replaces *lwe*(u) by $weight(e(u,v))$ if $weight(e(u,v)) \leq$ *lwe* (u).

   It can be seen that once every processor has processed its edges, *lwe*(u) will be the weight of the least weight edge incident from any sampled cluster. To identify the corresponding edge e_u, each processor scans each of its edges and compare their weight with *lwe*(u). The edge is then added to the spanner and its *live* variable is set to false.

3. *Selecting edges E_s^i for spanner*

   (a) For each *live* vertex $u \in V$, the processors P_u compute a perfect hash-function h_u for the cluster centers of the neighbors of u. In particular, $h_u : \{f_{\mathcal{C}_i}(v) | e(u,v) \in E'\} \longrightarrow \{1, 2, \cdots, n_u\}$.

   (b) The processors P_u copy *lwe*(u) into $A_u[i]$ for each $i \leq n_u$.

   (c) Each processor $p \in P_u$ processes its list l_p of live edges sequentially, and for each edge $e(u,v) \in l_p$ with $weight(e(u,v)) < A_u[h_u(v)]$, processor p replaces $A_u[h_u(v)]$ with $weight(e(u,v))$. (Note that all the edges incident on u from same cluster will write into same location.)

   It can be seen that once every processor has processed its edges, if c is a cluster centered at o and incident onto u with weight less than *lwe*(u), then $A_u[h_u(o)]$ stores the least weight edge of set $E'(u,c)$. The edge is then added to the spanner, and the set of edges $E'(u,c)$ is removed (by setting their *live* variable to false).

   The clustering $\mathcal{C}_i$ is defined as follows. Each vertex $u \in V'$ with *sampled*(u)=false does the following. If *lwe*(u)= ∞, then u becomes dead by setting its *live* variable to false, otherwise let v be the other endpoint of edge e_u. We replace *center*(u) by *center*(v).

4. *Deleting intra-cluster edges*

   Each processor $p \in P_u$ scans its live edges l_p, and for each edge $e(u,v)$ with *center*(u) = *center*(v), the variable *live*($e(u,v)$) is set to *false*.

Theorem 5.1 *Given a weighted graph with m edges, an integer k, and $\tau \geq \log^* n$, a $(2k-1)$-spanner of $O(kn^{1+1/k})$ expected size can be computed in priority CRCW PRAM model in $O(\tau)$ steps using $O(m)$ memory space and optimal work.*

6 Concluding Remarks

We described a simple linear time randomized algorithm for computing a $(2k-1)$-spanner of weighted graphs with optimal size given the validity of Erdős's *girth conjecture*. We have also shown that the algorithm, due to its local approach, can be implemented in external-memory and parallel computation environment with near optimal running times. The algorithm can be adapted to distributed environment as well where it would take constant (function of k) number of rounds and linear communication.

For unweighted graphs, we showed that our $(2k-1)$-spanner is indeed a $(\frac{3}{2}k, k-1)$-spanner, i.e., a spanner with reduced multiplicative stretch below $(2k-1)$ at the expense of increased additive stretch $k-1$. It would be nice to see further tradeoff in this direction while keeping the linear running time of the algorithm.

Another interesting problem would be to design efficient dynamic algorithms for maintaining sparse spanners for weighted graphs.

References

[1] N. Alon, S. Hoory, and N. Linial. The moore bound for irregular graphs. *Submitted for Journal publication*, 2000.

[2] 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.

[3] B. Awerbuch. Complexity of network synchronization. *Journal of Ass. Compt. Mach.*, pages 804–823, 1985.

[4] 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.

[5] H. Bast and T. Hagerup. Fast and reliable parallel hashing. In *Proceedings of 3rd Annual ACM Symposium on Parallel Algorithms and Architecture (SPAA)*, pages 50–61, 1991.

[6] B. Bollobás. In *Extremal Graph Theory*, page 164. Academic Press, 1978.

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

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

[9] 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.

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

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

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

[13] 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.

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

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

[16] U. Zwick. Personal communication.