A Fast Algorithm For Constructing Sparse Euclidean Spanners

(Extended Abstract)

Gautam Das [*][†] Giri Narasimhan[†]

Abstract

Let $G = (V, E)$ be a n-vertex connected graph with positive edge weights. A subgraph G' is a t-spanner if for all $u, v \in V$, the distance between u and v in the subgraph is at most t times the corresponding distance in G. We design an $O(n \log^2 n)$ time algorithm which, given a set V of n points in k-dimensional space, and any constant $t > 1$, produces a t-spanner of the complete Euclidean graph of V. This algorithm retains the spirit of a recent $O(n^3 \log n)$-time greedy algorithm which produces t-spanners with a small number of edges and a small total edge weight; we use graph clustering techniques to achieve a more efficient implementation. Our spanners have similar size and weight sparseness as those constructed by the greedy algorithm.

1 Introduction

Let $G = (V, E)$ be a n-vertex connected graph with positive edge weights. A subgraph G' is a t-spanner if for all $u, v \in V$, the distance between u and v in the subgraph is at most t times the corresponding distance in G. The value of t is known as the *stretch factor* of G'. Let V be a set of n points in k-dimensional space. An *Euclidean graph* has V as its vertices, and its edges are straight line segments joining pairs of points, with edge weights being their Euclidean distances. The *complete* Euclidean graph contains all $n(n-1)/2$ edges.

Although complete graphs represent ideal communication networks, they are expensive to build in practice, and sparse spanners represent low cost alternatives. Sparseness of spanners is usually measured by various criteria such as few edges, small weight, and small degree. Spanners for complete Euclidean graphs as well as for arbitrary weighted graphs find applications in robotics, network topology design, distributed systems, design of parallel machines, and have also been investigated by graph theorists. Most of the recent literature on spanners are referenced in [4, 9].

Let MST refer to a minimum spanning tree of the complete graph. For complete Euclidean graphs in 2-dimensions, an $O(n \log n)$-time algorithm is described in [7] which produces t-spanners with $O(n)$ edges and weight $O(wt(MST))$, which is optimal. The problem gets more difficult in higher dimensions. In k-dimensional space, there are several algorithms that run in $O(n \log n)$ time [8, 10]. However they only guarantee that the number of spanner edges is $O(n)$; the *total weight* could be arbitrarily large. In [4] an $O(n \log n)$ time algorithm is described which generates spanners in k-dimensions with weight $O(\log n) \cdot wt(MST)$. The nature of this algorithm is such that the $O(\log n)$ factor cannot be eliminated, so the algorithm always produces a suboptimally weighted spanner.

In contrast, in [1, 4, 5] a simple ***greedy algorithm*** is presented which generates spanners with $O(n)$ edges ***and*** a weight which is likely to be asymptotically optimal for all dimensions. In [4] it is shown that for any dimension k the weight is $O(\log n) \cdot wt(MST)$, and in [5] it is shown that for $k \leq 3$ the weight is $O(wt(MST))$. In [5] it is conjectured that the weight is actually asymptotically optimal in any dimension and that the $O(\log n)$ factor can be elim-

[*]Supported in part by NSF grant CCR-930-6822

[†]Math Sciences Dept., Memphis State University, Memphis, TN 38152. *e-mail:* dasg/giri@next1.msci.memst.edu

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association of Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

inated by a more careful analysis. The constants implicit in the O-notation depend only on t and k. The greedy algorithm has also been used to generate spanners of arbitrary weighted graphs, and there again, it is superior to other algorithms in the sense that it produces spanners with very small weight.

However, the greedy algorithm is handicapped by a slow running time, mainly because it has to make a large number of shortest path queries. In the Euclidean case, the best-known implementation has a running time of $O(n^3 \log n)$. In this paper, we design an $O(n \log^2 n)$-time algorithm to construct t-spanners, which retains the spirit of the greedy algorithm. This new algorithm is faster because we use ***graph clustering*** techniques to answer shortest path queries only approximately. The spanners produced by this algorithm have the same asymptotic size and weight bounds as the spanners produced by the greedy algorithm. Furthermore, any improvements in these bounds for the greedy algorithm spanners is likely to lead to similar improvements in the bounds for the spanners produced by this algorithm. Graph clustering is not new, and in fact there are several algorithms which use a variety of clustering techniques to generate spanners of Euclidean as well as arbitrary weighted graphs [2, 3, 6, 8, 10]. What is interesting is the way in which we exploit a simple clustering technique in our implementation.

In the next section we introduce notation, summarize previous relevant research, and give an overview of our algorithm. Section 3 describes the clustering technique used. Section 4 describes the algorithm. We conclude with some open problems.

2 Preliminaries

Let $G = (V, E)$ be a n-vertex connected graph with positive edge weights. The weight of a single edge e is $wt(e)$, the weight of a set of edges E' is $wt(E')$, and the weight of the entire graph is $wt(G)$. Let $sp_G(u, v)$ be the weight of the shortest path in G between u and v. Let $d(u, v)$ refer to the Euclidean distance between points u and v in Euclidean space.

In [1, 4], a *greedy algorithm* is developed for constructing t-spanners of arbitrary weighted graphs, for any $t > 1$. We reproduce the algorithm in Figure 1. Let us apply this algorithm to a complete Euclidean graph in k-dimensional space and let the output graph be $G' = (V, E')$.

It is easy to see that the output is a t-spanner, since for any (u, v) that is not an edge of G', $sp_{G'}(u, v) \le t \cdot d(u, v)$. *In addition,* for any (u, v)

```
Algorithm GREEDY($G = (V, E), t$)
begin
order $E$ by non-decreasing weights
$E' \leftarrow \emptyset$, $G' \leftarrow (V, E')$
for each edge $e = (u, v) \in E'$ do
    if $sp_{G'}(u, v) > t \cdot wt(e)$ then
        $E' \leftarrow E' \cup \{e\}$, $G' \leftarrow (V, E')$
output $G'$
end.
```

Figure 1: The greedy algorithm

that is an edge of G', the weight of the second shortest path between u and v in G' is greater than $t \cdot d(u, v)$. The second property is crucial in showing that the total weight of the spanner is small. In [4] it is shown that G' has $O(n)$ edges and has a total edge weight of $O(\log n) \cdot wt(MST)$, where MST is a minimum spanning tree of V. In [5] it is shown that for $k \le 3$, the greedy algorithm outputs a t-spanner of weight $O(wt(MST))$. In that paper it is conjectured that this result is true for k-dimensions.

In this paper, our algorithm mimics the greedy algorithm and constructs a t-spanner G' with asymptotically same size and weight sparseness. Apart from the very short edges of the spanner, all other edges (u, v) satisfy the property that the weight of the second shortest path between u and v in G' is greater than $c \cdot d(u, v)$, where c is a constant dependent on t. Thus, techniques similar to those used in [4, 5] can be used to prove the same sparseness bounds. Details appear later.

The following concept will be useful in later sections. We define a certain kind of "partial" spanners of complete Euclidean graphs. Let V be a set of n points in k-dimensional space, $t > 1$ and $W > 0$. A graph G' is a (t, W)-spanner if for all $u, v \in V$ such that $d(u, v) \le W$, $sp_{G'}(u, v) \le t \cdot d(u, v)$. In other words, G' provides short paths between pairs of points whose inter-distance is at most W. For instance, at any stage of the greedy algorithm, if the edge most recently examined has weight L, the partially constructed spanner is a (t, L)-spanner.

2.1 Overview of the algorithm

We first run the $O(n \log n)$ time algorithm in [8] to get a graph $G = (V, E)$, which is a $\sqrt{t}$-spanner of the complete Euclidean graph. Although it may have a large weight, it has $O(n)$ edges. We shall then find a

$\sqrt{t}$-spanner G' of G. Clearly, G' will be a t-spanner of the complete graph.

If we were to run the original greedy algorithm on G to get G', it will have to answer $O(n)$ shortest path queries on G'. Our algorithm does better by speeding up the queries. We group the edges in E into $O(\log n)$ groups, such that each group has edges that are more or less of equal weight. Then we examine the edges, going from one group to the next in order of increasing weight, and at the same time building a partially constructed spanner, G'.

Immediately before processing a group of edges (of weight approximately W), the algorithm creates a *cluster-graph* H as follows. Roughly speaking, the vertices of G' are covered by a set of *clusters* such that each cluster consists of vertices that are within $O(W)$ distance from each other in G'. Each cluster is collapsed into a representative vertex, namely its *center*, to form a single macro vertex of H. These macro vertices are connected by edges of weight $\Theta(W)$ only if the distance between their centers in G' is $\Theta(W)$. Notice that all edge weights of H are more or less the same. Thus distances in G' are approximated by distances in H. A crucial observation, as proved later, is that in the Euclidean case the maximum degree of H turns out to be a constant. Thus the shortest path query can be answered in *constant time* as follows. Start with a macro vertex whose corresponding cluster contains u, and see whether v belongs to any of the clusters that are within $O(\sqrt{t})$ links from the start cluster. Since the degree is constant, and t is a constant, we need only explore a constant-sized portion of H. Once we have examined all the edges in a group, we create a cluster-graph again (but with a larger W), and process the next group of edges.

The running time of of our algorithm is dominated by the periodic clustering necessary.

3 Clustering a graph

In this section we first consider arbitrary weighted graphs. Let $G' = (V, E')$ be an any positive weighted graph, and $W > 0$. A *cluster* C is a set of vertices, with one vertex v its *center*, such that for all $u \in V$, $sp_{G'}(u, v) \leq W$ implies that $u \in C$. W is the *radius* of the cluster. A *cluster-cover* is a set of clusters $C_1, C_2, \ldots, C_m$, (with corresponding centers $v_1, v_2, \ldots, v_m$) such that their union is V. Clusters in a cluster-cover may overlap, however a cluster center does not belong to any other cluster.

We describe an obvious algorithm to compute a cluster-cover. First consider a procedure called SINGLE-SOURCE(G', v, W), which takes as input any weighted graph G', a start vertex v, and $W > 0$. It outputs all vertices u such that $sp_{G'}(v, u) \leq W$. It can be implemented as Dijkstra's single-source shortest path algorithm by using a heap and ensuring that the algorithm never visits vertices further than W from v. We now design a procedure CLUSTER-COVER(G', W) as follows. Select any vertex v_1 of G', and run SINGLE-SOURCE(G', v_1, W). The output is a cluster C_1 with center v_1. Then select any vertex v_2 not yet visited, and run SINGLE-SOURCE(G', v_2, W). The output is C_2 with center v_2. Continue this process until all vertices are visited. For each vertex, the procedure also computes the clusters to which it belongs as well as its distance from their respective centers.

Let $G' = (V, E')$ be any weighted graph, $W > 0$ and $0 < \delta < 1/2$. Assume a cluster-cover has been constructed with cluster radius δW. A *cluster-graph* H is defined as follows. The vertex set is V. The are two kinds of edges, *intra-cluster edges* and *inter-cluster edges*. For all C_i, and for all $u \in C_i$, $[u, v_i]$ is an intra-cluster edge (we use square brackets to distinguish cluster-graph edges from the edges of G'). Inter-cluster edges are between cluster centers. $[v_i, v_j]$ is an inter-cluster edge if, either $sp_{G'}(v_i, v_j) \leq W$, or there exists $e = (u, v) \in E'$ such that $u \in C_i, v \in C_j$. Cluster-graph edges have weights, and $wt([u, v])$ is defined as $sp_{G'}(u, v)$.

We now describe a procedure called CLUSTER-GRAPH(G', δ, W). It first calls CLUSTER-COVER($G', \delta W$). In fact, during this process all intra-cluster edges (and their weights) will also be computed. Recall that inter-cluster edges have to satisfy one of two conditions. Inter-cluster edges satisfying the first condition may be computed by running SINGLE-SOURCE on G' from each cluster center v_i and checking which other cluster centers are no further than W from v_i. The remaining inter-cluster edges may be computed as follows. For all $e = (u, v) \in E'$ do the following. For all cluster centers v_i, v_j such that u belongs to v_i's cluster and v belongs to v_j's cluster, add the inter-cluster edge $[v_i, v_j]$ with weight defined to be $wt([v_i, u]) + wt(e) + wt([v, v_j])$ (unless the inter-cluster edge already existed with a smaller weight).

The motivation for the cluster-graph is that it is a simpler structure than G'. Shortest path queries in G' can be translated into shortest path queries in H, and the latter can be performed more efficiently. The following two lemmas are obvious.

Lemma 3.1 *Intra-cluster edges are no larger that δW. Inter-cluster edges are larger than δW, but no larger than* $\max\{W, D + 2\delta W\}$*, where D is the weight of the largest edge in G'.*

Lemma 3.2 *Let a path between u and v in H have weight L. Then there is a path between u and v in G' with weight at most L.*

The next lemma is the converse, which is harder. We first introduce some definitions. A vertex u is defined to be ***sufficiently far*** from vertex v if, (1) no single cluster contains both, and (2) $sp_{G'}(u, v) > W - 2\delta W$. Define a ***cluster-path*** in H to be a path where the first and last edges are intra-cluster edges, but all intermediate edges are inter-cluster edges.

Lemma 3.3 *Let u be sufficiently far from v. Let L_1 be the weight of a path between u and v in G'. Then there exists a cluster-path between u and v in H with weight L_2 such that*

$$L_2 \leq \left(\frac{1+6\delta}{1-2\delta}\right) \cdot L_1$$

Proof : This lemma says that H approximates paths in G' only for pairs of vertices which are sufficiently far apart. Secondly, notice that the quality of the approximation depends on δ; smaller values of δ make paths in H closer in weight to paths in G', although H becomes denser.

Let the path from u to v having weight L_1 in G' be P. We shall use the notation $P(w, x)$ to denote the vertices of P between vertices w and x, not including w. We shall construct a cluster-path Q from u to v in H with weight L_2 as follows. follows. Let C_0 be any cluster (with center v_0) containing u. The first edge of Q is the intra-cluster edge $[u, v_0]$. Next, among all clusters with centers adjacent to v_0 in H, let C_1 (with center v_1) intersect the furthest vertex along $P(u, v)$, say w_1. Add the inter-cluster edge $[v_0, v_1]$ to Q. Next, among all clusters with centers adjacent to v_1 in H, let C_2 (with center v_2) intersect the furthest vertex along $P(w_1, v)$, say w_2. Add the inter-cluster edge $[v_1, v_2]$ to Q. This process may be continued until we reach a cluster center, v_m, whose cluster contains v. At this stage, complete Q by adding the intra-cluster edge $[v_m, v]$. Figure 2 illustrates some of this notions.

Case 1: $m = 1$. In this case there is only one inter-cluster edge along Q. Since u is sufficiently far from v, we know that

$$L_1 > W - 2\delta W$$

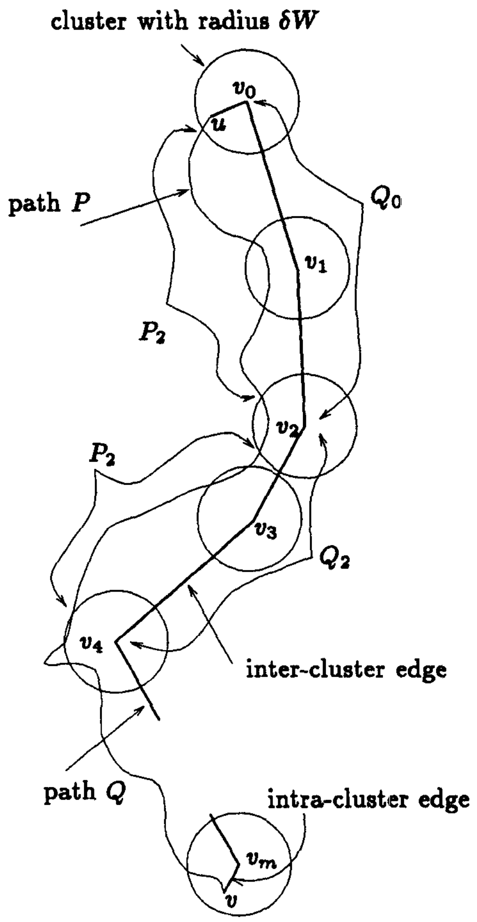

Figure 2: Paths in G' and H

Now $L_2 = wt([u, v_0]) + wt([v_0, v_1]) + wt([v_1, v])$. But $wt([u, v_0])$ and $wt([v_1, v])$ are each at most δW, while $wt([v_0, v_1]) = sp_{G'}(v_0, v_1) \leq sp_{G'}(v_0, u) + sp_{G'}(u, v) + sp_{G'}(v, v_1) \leq 2\delta W + L_1$. Thus

$$L_2 \leq L_1 + 4\delta W$$

We now have two inequalities relating L_1, L_2 and W. After eliminating W, we get

$$L_2 < \left(\frac{1+2\delta}{1-2\delta}\right) \cdot L_1$$

Case 2: $m \geq 2$ and is even. In this case, suppose $[v_i, v_{i+1}]$ and $[v_{i+1}, v_{i+2}]$ are any two consecutive inter-cluster edges on Q. We observe that the sum of their weights is greater than W. If this were not so, then the edge $[v_i, v_{i+2}]$ (which exists by the definition of the cluster-graph) would have instead been added to Q by the above process.

We divide Q into portions $Q_0, Q_2, \ldots$, where Q_{2i} is the portion between v_{2i} and v_{2i+2}. Similarly, we divide P into portions $P_0, P_2, \ldots$, where P_{2i} is the portion between the last vertex intersecting C_{2i} and the first vertex intersecting C_{2i+2} (see Figure 2). We shall first prove that for any even i, the weight of Q_{2i} is no more than a constant (which depends upon δ) times the weight of P_{2i}.

Let the weight of P_{2i} be p_{2i} and that of Q_{2i} be q_{2i}. Since there cannot be an inter-cluster edge between v_{2i} and v_{2i+2}, we have

$$p_{2i} > W - 2\delta W$$

Select r to be any vertex of P_{2i} within the intermediate cluster C_{2i+1}. The vertex r splits P_{2i} into two portions. Let p'_{2i} (p''_{2i}) be the weight of the initial (final) portions; thus $p_{2i} = p'_{2i} + p''_{2i}$. So $wt([v_{2i}, v_{2i+1}]) \leq p'_{2i} + 2\delta W$, and similarly $wt([v_{2i+1}, v_{2i+2}]) \leq p''_{2i} + 2\delta W$. Adding the two, we get

$$q_{2i} \leq p_{2i} + 4\delta W$$

We now have two inequalities relating p_{2i}, q_{2i} and W. After eliminating W, we get

$$q_{2i} < \left(\frac{1+2\delta}{1-2\delta}\right) \cdot p_{2i}$$

Summing over all even values of i, and taking into account the two intra-cluster edges at either ends of Q, we get

$$L_2 < \left(\frac{1+2\delta}{1-2\delta}\right) \cdot L_1 + 2\delta W$$

Since u is sufficiently far from v, we know that $L_1 > W - 2\delta W$. That is, $L_1 \cdot 2\delta/(1-2\delta) > 2\delta W$. Substituting for $2\delta W$ in the above inequality and simplifying, we get

$$L_2 < \left(\frac{1+4\delta}{1-2\delta}\right) \cdot L_1$$

Case 3: $m \geq 3$ and is odd. In this case, the analysis will be exactly the same as in Case 2, except that we have to account for the last inter-cluster edge along Q and correspondingly the portion of P between the last two clusters. Let q_{m-1} be the weight of $[v_{m-1}, v_m]$, and let p_{m-1} be the weight of the portion of P between the last vertex intersecting C_{m-1} and first vertex intersecting C_m. Clearly $q_{m-1} \leq p_{m-1} + 2\delta W$. The inequality does not change if we rewrite it as $q_{m-1} \leq \left(\frac{1+2\delta}{1-2\delta}\right) \cdot p_{m-1} + 2\delta W$. We then sum up the p's and q's as in Case 2, and get

$$L_2 < \left(\frac{1+2\delta}{1-2\delta}\right) \cdot L_1 + 4\delta W$$

Since $L_1 > W - 2\delta W$, we get $L_1 \cdot 4\delta/(1-2\delta) > 4\delta W$. Substituting for $4\delta W$ in the above inequality and simplifying, we get

$$L_2 < \left(\frac{1+6\delta}{1-2\delta}\right) \cdot L_1$$

The constant in Case 3 dominates, which proves the lemma. □

3.1 Clustering of partial Euclidean spanners

Although described for arbitrary weighted graphs, in our algorithm we will always be computing cluster-graphs for partial spanners of complete Euclidean graphs under the following conditions. Let $t > 1$, $\alpha > 0$, $0 < \delta < 1/2$, $\theta > 0$, and $W > 0$. For a set V of n points in k-dimensional space, let G' be a $(t, \alpha W)$-spanner such that all its edges are smaller than θW. Run CLUSTER-GRAPH(G', δ, W), and let H be the computed cluster-graph. H has several additional properties, which are used in analyzing the running time of our algorithm.

Lemma 3.4 *Let $\beta > 0$ be any constant. Inside any sphere of radius βW there can be at most a constant (which depends upon α, β, δ, t, and k) number of cluster centers.*

Proof : We first provide a lower bound on the spatial separation between any pair of cluster centers. Let v_i, v_j be two cluster centers, and let $d(v_i, v_j) = L$. There are two cases, either $L > \alpha W$ or $L \leq \alpha W$. If it is the latter case, then $sp_{G'}(v_i, v_j) \leq tL$. But $sp_{G'}(v_i, v_j) > \delta W$ (Lemma 3.1), hence $L > \delta W/t$. So in either case, $L > \min\{\alpha W, \delta W/t\}$.

Given a sphere of radius βW, we can pack at most a constant (which depends upon α, β, δ, t, and k) number of points inside it, such that the spatial separation between any pair of points is greater than $\min\{\alpha W, \delta W/t\}$. □

Lemma 3.5 *A vertex belongs to at most a constant (which depends upon α, δ, t, and k) number of clusters.*

Proof : Consider any vertex u. Let C be a cluster containing u, with cluster center v. $d(u,v) \leq sp_{G'}(u,v) \leq \delta W$. Consider a sphere of radius δW centered at u. The cluster centers of all clusters containing u have to lie within the sphere. Invoking Lemma 3.4 completes the proof. □

Lemma 3.6 *The subgraph of H induced by the inter-cluster edges has at most a constant degree (which depends α, δ, θ, t, and k).*

Proof : Let $[v_i, v_j]$ be an inter-cluster edge. By Lemma 3.1, $d(v_i,v_j) \leq sp_{G'}(v_i,v_j) \leq \max\{W, \theta W + 2\delta W\}$. Consider a sphere of radius $\max\{W, \theta W + 2\delta W\}$ centered at v_i. The cluster centers adjacent to v_i have to lie within the sphere. Invoking Lemma 3.4 completes the proof. □

We introduce one more procedure to be used by the algorithm. Let G' be a $(t, \alpha W)$-spanner satisfying the conditions laid out earlier, and let H be its cluster-graph. The procedure CHECK-PATH(H, u, v, L) returns "true" if there is a cluster-path from u to v in H with weight at most L, and "false" otherwise.

This procedure is always called by our spanner algorithm under fortunate circumstances. There is a constant $\gamma > 0$ such that, the input values of L are always at most γW. Because of this restriction, the procedure can be made to run in *constant time*. To see this, consider initiating from u a brute-force search for such a cluster-path. There are at most a constant number of intra-cluster edges leading from u to different cluster centers. The inter-cluster edges have each weight at least δW, thus if such a cluster-path exists, it can have at most γ/δ inter-cluster edges. We also know that the subgraph of H induced by inter-cluster edges has a constant degree (Lemma 3.6). We can conclude that the brute-force search initiated from u will only have to examine a *constant sized* subgraph of H, and verify whether v belongs to this subgraph, and if so, whether there exists a cluster-path from u to v of weight at most L.

This procedure is used in lieu of the time consuming shortest path queries of the original greedy algorithm.

```
Algorithm FAST-GREEDY($V, t$)
begin
$\delta \leftarrow \frac{1}{2}\left(\frac{\sqrt[4]{t}-1}{\sqrt[4]{t}+3}\right)$
use algorithm in [8], create a $\sqrt{t}$-spanner $G = (V, E)$
order $E$ by non-decreasing weight
let the largest edge in $E$ have weight $D$
create the intervals
        $I_0 = (0, D/n]$,
        $I_i = (2^{(i-1)}D/n, 2^i D/n]$ for $i = 1, 2, \ldots, \log n$
let $E_i$ be the (sorted) edges of $E$ with weights in $I_i$
$E' \leftarrow E_0$, $G' \leftarrow (V, E')$
for $i \leftarrow 1$ to $\log n$ do
        $W_i \leftarrow 2^{(i-1)}D/n$
        $H \leftarrow$ CLUSTER-GRAPH($G', \delta, W_i$)
        for each edge $e \in E_i$ do
                EXAMINE-EDGE($e$)
output $G'$
end.
```

Figure 3: An $O(n \log^2 n)$-time algorithm

4 The spanner algorithm

In this section we illustrate an $O(n \log^2 n)$ time algorithm for constructing t-spanners of complete Euclidean graphs in k-dimensional space. The algorithm is shown in Figure 3, and one of its subroutines is shown in Figure 4. As one can see, it retains the spirit of the greedy algorithm, except for the graph clusterings.

4.1 Analysis of the algorithm

We analyze three aspects of the FAST-GREEDY algorithm. First, we prove that G' is indeed t-spanner. We then show that the spanner has a small weight. Finally we show that a proper implementation runs in $O(n \log^2 n)$ time.

Lemma 4.1 *The graph G' produced by the* FAST-GREEDY *algorithm is a t-spanner of the complete Euclidean graph.*

Proof : To see that G' is a t-spanner, consider any edge $e = (u, v)$ of E which is not added to E' in procedure EXAMINE-EDGE. A cluster-path (in fact, any path) from u to v in the cluster-graph corresponds to an equivalent path in G' with the same weight (Lemma 3.2). Thus e is discarded if $sp_{G'}(u,v) \leq \sqrt{t} \cdot d(u,v)$, which implies that G' is a $\sqrt{t}$-spanner of G, and is thus a t-spanner of the complete graph. □

Let us now estimate the weight of the spanner. The E_0 edges do not contribute much because their

```
Algorithm EXAMINE-EDGE($e = (u, v)$)
begin
if not CHECK-PATH($H, u, v, \sqrt{t} \cdot d(u, v)$) then
        $E' \leftarrow E' \cup \{e\}$, $G' \leftarrow (V, E')$
        for all cluster centers $w, x$ such that
                    $u$ is in $w$'s cluster and
                    $v$ is in $x$'s cluster
        do
                add inter-cluster edge $[w, x]$ to $H$
                $wt([w, x]) \leftarrow wt([w, u]) + d(u, v) +$
                    $wt([v, x])$
end.
```

Figure 4: Deciding whether an edge should be added

total weight is at most D, and $D \leq wt(MST)$. We shall estimate $wt(E' \setminus E_0)$.

Lemma 4.2 *Let $e = (u, v) \in E' \setminus E_0$. Then the weight of the second shortest path between u and v in G' is greater than $\sqrt[4]{t} \cdot d(u, v)$.*

Proof : Let C be the shortest simple cycle in G' containing e. We have to estimate $wt(C) - d(u, v)$. Let $e_1 = (u_1, v_1)$ be the largest edge on the cycle. Then $e_1 \in E \setminus E_0$, and among the cycle edges it is examined last by the algorithm. Let us consider the scenario while the algorithm is examining e_1.

There is an alternate path in G' from u_1 to v_1 of weight $wt(C) - d(u_1, v_1)$. But since the algorithm eventually decides to add e_1 to the spanner, at that moment the weight of each cluster-path from u_1 to v_1 is larger than $\sqrt{t} \cdot d(u_1, v_1)$. Notice that $wt(u_1, v_1)$ is larger than W_i, where δW_i is the current cluster radius. This implies that u_1 and v_1 are not contained in any one cluster. Thus u_1 is sufficiently far from v_1. By Lemma 3.3 this implies that the weight of each path in G' from u_1 to v_1, in particular $wt(C) - d(u_1, v_1)$, is larger than $\sqrt{t} \cdot \left(\frac{1-2\delta}{1+6\delta}\right) \cdot d(u_1, v_1)$. Substituting the value of δ selected by the algorithm, this simplifies to $\sqrt[4]{t} \cdot d(u_1, v_1)$. Since $wt(C) - d(u, v) \geq wt(C) - d(u_1, v_1)$, the lemma is proved. □

Lemmas 4.1 and 4.2 are crucial in proving that $E' \setminus E_0$ has a small weight. Using techniques almost identical to those in [4, 5], we can derive a bound on the weight of $E' \setminus E_0$, which is asymptotically the same as the weight of the spanner produced by the greedy algorithm. We omit details from this version of the paper.

We now analyze the running time of the algorithm.

Lemma 4.3 *At any stage, if the most recently examined edge has weight L, at that stage G' is a $(t, L/t)$-spanner.*

Proof : Consider in general any t-spanner G' and let u and v be any pair of vertices. Then $sp_{G'}(u, v) \leq t \cdot d(u, v)$, which implies that any edge on this path has weight at most $t \cdot d(u, v)$. Thus in our case, the edges to be examined in the future (weights larger than L) cannot contribute to short spanner paths between points whose spatial separation is at most L/t. □

The above lemma implies that each cluster-graph H generated by the CLUSTER-GRAPH procedure (prior to the examination of edges in E_i) satisfies all the properties and lemmas discussed in Section 3.1. During the processing of E_i, new inter-cluster edges may be added by the EXAMINE-EDGE procedure, but H still satisfies those properties.

Lemma 4.4 FAST-GREEDY *runs in $O(n \log^2 n)$ time.*

Proof : The initial stages of the algorithm involve a call to the spanner algorithm in [8], and sorting, which together take $O(n \log n)$ time.

Consider EXAMINE-EDGE. As seen earlier, a call to CHECK-PATH takes constant time. Adding the new inter-cluster edges takes constant time, because there are only a constant number of clusters containing either of the end points of the processed edge. Thus each call to EXAMINE-EDGE takes constant time. EXAMINE-EDGE is itself called $O(n)$ times.

Consider CLUSTER-GRAPH. This procedure first calls CLUSTER-COVER. On partial spanners, the latter can be made to run in $O(n \log n)$ time because there are $O(n)$ edges and each vertex is visited by the SINGLE-SOURCE procedure at most a constant number of times (Lemma 3.5), thus there are overall $O(n)$ heap operations, each taking $O(\log n)$ time. The CLUSTER-COVER procedure can simultaneously build the intra-cluster edges of the cluster-graph. Inter-cluster edges are built in two stages. In the first stage, the SINGLE-SOURCE procedure is run from every cluster center, and in a manner similar to the proof of Lemma 3.5, we can show that each vertex is visited at most a constant number of times. This stage therefore takes $O(n \log n)$ time. In the second stage, each edge (u, v) of G' is checked; there are only a constant number of clusters containing u or v, thus this stage can be processed in linear time. Thus each call to CLUSTER-GRAPH takes $O(n \log n)$ time. CLUSTER-GRAPH is itself called $\log n$ times.

Hence the time complexity of FAST-GREEDY is $O(n \log^2 n)$. □

5 Conclusions

In this paper we show how a simple and natural graph clustering technique can be effectively used in speeding up an existing greedy algorithm for constructing Euclidean graph spanners. We conclude by listing some open problems.

1. Is it possible to reduce the running time to $O(n \log n)$? This may require doing some sort of clustering which *exploits* earlier clusterings. One problem with such an approach is that the error introduced in path length estimations (Lemma 3.3) may multiply beyond a constant factor.

2. The original greedy algorithm has been successful in producing small weight spanners for arbitrary weighted graphs [4]. There exist other spanner algorithms for general graphs which are faster, but none produce spanners with smaller weight. It would be interesting if we can efficiently implement the greedy algorithm for arbitrary weighted graphs. Of course, we would not be able to exploit several nice properties which arise in the geometric case, such as constant degree cluster-graphs.

3. The conjecture in [5] that for any dimensions the weight of the spanner produced by the greedy algorithm is $O(wt(MST))$ needs to be resolved. If proven true, it is likely to imply that asymptotically optimally weighted spanners in any dimensions can be computed in $O(n \log^2 n)$ time.

4. The clustering technique used in this paper is simple and natural; it would interesting to see whether it has other applications in problems involving geometric graphs, especially in speeding up existing algorithms.

References

[1] I. Althöfer, G. Das, D.P. Dobkin, D. Joseph, J. Soares: On Sparse Spanners of Weighted Graphs: Discrete and Computational Geometry, 9, 1993, pp. 81-100.

[2] B. Awerbuch, D. Peleg: Sparse Partitions: IEEE Foundations of Computer Science, 1993.

[3] E. Cohen: Fast Algorithms for Constructing t-Spanners and Paths with Stretch t: IEEE Foundations of Computer Science, 1993.

[4] B. Chandra, G. Das, G. Narasimhan, J. Soares: New Sparseness Results on Graph Spanners: to appear in International Journal of Computational Geometry and Applications.

[5] G. Das, P. Heffernan, G. Narasimhan: Optimally Sparse Spanners in 3-Dimensional Euclidean Space: ACM Symposium on Computational Geometry, 1993, pp. 53-62.

[6] G. Das, P. Heffernan: Constructing Degree-3 Spanners with Other Sparseness Properties: International Symposium on Algorithms and Computations, 1993.

[7] C. Levcopoulos, A. Lingas: There are Planar Graphs Almost as Good as the Complete Graphs and as Short as the Minimum Spanning Trees. Symposium on Optimal Algorithms, LNCS, Springer-Verlag, 1989, pp. 9-13.

[8] J.S. Salowe: Construction of Multidimensional Spanner Graphs with Applications to Minimum Spanning Trees: ACM Symposium on Computational Geometry, 1991, pp. 256-261.

[9] J. Soares: Graph Spanners: Ph.D Thesis, Univ. of Chicago Technical Report CS 92-14, 1992.

[10] P.M. Vaidya: A Sparse Graph Almost as Good as the Complete Graph on Points in K Dimensions: Discrete and Computational Geometry, 6, 1991, pp. 369-381