Journal of Algorithms and Computation

journal homepage: http://jac.ut.ac.ir

α-Gap Greedy Spanner

Hosein Salami*[1] and Mostafa Nouri-Baygi†[2]

[1,2]Department of Computer Engineering, Ferdowsi University of Mashhad

ABSTRACT

In this paper, we have introduced a new geometric spanner called α-Gap greedy spanner, which is a parametric approximation of the well-known Gap-greedy spanner. We will show theoretically and experimentally that this spanner is similar to the Gap-greedy spanner in terms of qualitative features such as weight and maximum degree of vertices. Two algorithms have been proposed with running time $O(n^2 \log n)$ for constructing the α-Gap greedy spanner. Space complexity of the first algorithm is $O(n^2)$, but it is reduced to $O(n)$ in the second one. Also, we have shown on the points placed randomly in a unit square, the α-Gap greedy spanner can be constructed in the expected $O(n \log n)$ time.

ARTICLE INFO

Article history:
Research Paper
Received 12, December 2020
Received in revised form 14, April 2021
Accepted 11 May 2021
Available online 01, June 2021

Keyword: computational geometry, geometric spanners, gap greedy spanner, construction algorithms, algorithm complexity.

AMS subject Classification: 68U05, 05C85, 97K30, 68Q25.

1 Introduction

A geometric network on a set V of n points on the plane is the weighted graph $G = (V, E)$ where the weight of each $(p, q) \in E$ is equal to the Euclidean distance $|pq|$ between the endpoints of the edge. Assume $t > 1$ is a real number, a (directed) geometric t-spanner

*hossein.salami@mail.um.ac.ir
†Corresponding author: M. Nouri-Baygi. Email: nouribaygi@um.ac.ir

on V is a sub-graph G' of the complete geometric network on V if, for every pair of points p and q, a (directed) path in G' between p and q of maximum weight $t|pq|$ exists. We call this path a *t-path* between p and q.

One of the challenging topics for researchers is to create low-cost methods (in terms of time and space complexity) to build high-quality spanners (based on some qualitative criteria such as small edge size (being lightweight), bounded degree of each vertex, ...). These goals are contradictory because in most cases creating high-quality spanners imposes a high cost. For situations where reducing construction costs is more important, the use of low-cost methods that provide an approximation of the desired spanner is suggested [9, 11, 5].

A graph G is called θ-angle-constrained, if any two distinct edges in G sharing an endpoint make an angle of at least θ [8]. A θ-angle-constrained spanner is useful for a wireless network that uses *directional antennas*, because in addition to ensuring a reduction in signal interference and energy consumption, it provides short distances between each pair of the network nodes [12]. Carmi and Smid [8] proposed an algorithm that takes $O(n \log n)$ time to construct a θ-constrained spanner on a set of points, but they did not examine qualitative features of the resulted spanner. *Path-greedy* and *Gap-greedy* spanners are two well-known geometric spanners that have been shown to be angle-constrained [8, 14]. The advantage of using these spanners is that their quality is known, however, the algorithms for computing these spanners impose high costs.

PathGreedy algorithm, which is the most well-known algorithm for computing the Path-greedy spanner, runs in near-cubic time. Das and Narasimhan [9] proposed *Approximate-Greedy* algorithm that produces a low-cost approximation of this spanner in $O(n \log^2 n)$ time. Gudmundsson et al. [11] made some changes in order to reduce time complexity of the construction to $O(n \log n)$. Although ApproximateGreedy algorithm results in a spanner with theoretical features similar to the Path-greedy spanner, in practice it performs worse than expected in most cases [10]. In contrast to these algorithms, which provide a non-parametric approximation of the Path-greedy spanner, Bar-On and Carmi [5] proposed an algorithm that constructs a parametric approximation of this spanner. Their proposed algorithm, which has a time complexity of $O(n^2 \log n)$, has a parameter which, when equated with input stretch factor, produces the same output as PathGreedy algorithm.

Note that other improvements have been made to PathGreedy algorithm, including algorithm of Bose et al. [6], which has time and space complexity of $O(n^2 \log n)$ and $O(n^2)$, respectively. Alewijnse et al. [2], proposed a linear space algorithm that creates this spanner using *well-separated pairwise decomposition*. They proved that their algorithm can compute in $O(n \log^2 n \log^2 \log n)$ expected time the Path-greedy spanner on a point set whose points are uniformly distributed in a $\sqrt{n} \times \sqrt{n}$ square. However, on an arbitrary point set, time complexity of their algorithm is $O(n^2 \log^2 n)$.

The output produced by the *GapGreedy* algorithm, which is proposed by Aria and Smid [3] to compute the Gap-greedy spanner, provides an angle-constrained spanner. GapGreedy algorithm imposes a high cost, i.e., it takes $O(n^3)$ time and $O(n^2)$ space for computing the Gap-greedy spanner, so the same authors proposed another algorithm which is called *FastGapGreedy*. FastGapGreedy computes in $O(n^2 \log n)$ time a spanner that is asymp-

totically similar to the Gap-greedy spanner, but not necessarily identical and specifically not an angle-constrained.
Bakhshesh and Farshi have proposed two algorithms to compute the Gap-greedy spanner [4]. Their algorithms, that use the well-separated pairs decomposition, take $O(n^3)$ and $O(n^2)$ time to compute this spanner, respectively. Space complexity of their algorithms are also $O(n)$ and $O(n^2)$, respectively.
In this paper, we propose algorithms that compute a parametric approximation of the Gap-greedy spanner. The approximated spanner, which we call α-*Gap greedy* spanner, is constructed at a lower cost than the Gap-greedy spanner. The proposed algorithms have a parameter called α through which the difference between the α-Gap greedy spanner and the Gap-greedy spanner can be determined.
The rest of the paper is organized as follows. In Section 2, the Gap property and the Gap-greedy spanner are introduced. In Section 3, the α-Gap greedy spanner, its construction algorithms, an analysis of the proposed algorithms and an examination of quality of resulted spanner are presented. Section 4 is devoted to the construction of the α-Gap greedy spanner on the points placed randomly in a unit square. In Section 5, results of experiments done on the proposed algorithms are presented.

2 Gap property and the Gap-greedy spanner

In the following sections, we use some notations defined in [13]. We use angle(pq, rs), for the angle between two directed edges (p, q) and (r, s), which is defined as the angle of translation of (p, q) and (r, s) such that their sources are at the origin. Let $w \geq 0$ be a real number, and let E be a set of directed edges in $\mathbb{R}^d$. We say that E satisfies the w-gap property if for any two distinct edges (p, q) and (r, s) in E, we have

$$|pr| > w \cdot \min(|pq|, |rs|)$$

If, in addition to the above condition, the following condition holds for any two distinct edges (p, q) and (r, s) in E, then E satisfies the strong w-gap property.

$$|qs| > w \cdot \min(|pq|, |rs|)$$

Using the w-gap property, a way of examining the existence of a short path between two arbitrary points was proposed by Arya and Smid [3]. In this paper, a modified version that is proposed by Bakhshesh and Farshi [4] is used that is as follows. Let $G = (V, E)$ be a directed graph and assume $p, q \in V$ are two distinct points. If there exists an edge $(r, s) \in E$ such that (i) the vectors pq and rs have approximately the same direction, (ii) $|rs|$ is not larger than $|pq|$, and (iii) at least one of the distances $|pr|$ and $|qs|$ is small (relative to the length of (r, s)), then we can obtain a short path between p and q, by first going from p to r, then following the edge (r, s), and finally going from s to q. Lemma 2 explains the proposed scheme formally.
Lemma 1 [4] Let t, θ, and w be real numbers, such that $0 < \theta < \frac{\pi}{3}$, $0 \leq w < 1 - 2\sin(\frac{(\sin\theta/2)}{2})$, and $t \geq \frac{1}{(1-2\sin(\theta/2)-2w)}$. Let p, q, r, and s be points in $\mathbb{R}^d$, such that

1. $p \neq q$, $r \neq s$,
2. angle$(pq, rs) \leq \theta$,
3. $|rs| \leq |pq|$, and
4. $|pr| \leq w|rs|$.

Then $|pr| < |pq|$, $|sq| < |pq|$, and $t|pr| + |rs| + t|sq| \leq t|pq|$.
Note that condition (4) can also be extended to the case when $|qs| \leq w|rs|$.
Definition 1 Given real numbers t, θ, and w, and points p, q, r, and s satisfying the conditions of Lemma 2, we refer to the edge (r, s) as an *evident edge* for (p, q). Now we explain how *GapGreedy* algorithm construct the Gap-greedy spanner. The construction is started with an empty graph G, and then all pairs of points are processed in non-decreasing order of their distances. At the time when a point pair (p, q) is to be processed, if no evident edge is found for it in G, (p, q) is added to G as a new edge. Note that this search, in worst case, requires examining all edges of G.
However, according to Lemma 2, the evident edges of a point pair are not far from it, and therefore we can use this *locality of the evidences* feature to limit the search space. The algorithms presented in this paper use this feature to speed up the construction process. The proposed algorithms store some information in the form of a set of *cones* at every point, about the evident edges that are close to it. The cones are used by the algorithms as an alternative and less expensive way to determine the existence of evident edges for pairs of points.

3 α-Gap greedy Spanner

In this section, two algorithms are presented that compute the α-Gap greedy spanner on the input point set. In the algorithms presented, whenever an evident edge is found for a pair (p, q), some information will be stored in p so that the existence of an evident edge for the other pairs that are close to (p, q) can be examined less expensively. In the proposed algorithms, we use the concept of *α-evident edge*, which is defined as follows.
Definition 2 We call an edge (r, s) an α-evident edge for (p, q), if (r, s) is an evident edge for (p, q) and $angle(pq, rs) \leq \theta - \alpha$, where α is a real number such that $0 < \alpha < \theta$.
In the Gap-greedy spanner, the angle between each pair and its evident edge is at most θ, however, in the α-Gap greedy spanner, for some pairs this angle is deliberately considered to be less than θ (we discuss the reason later). This difference is determined by α parameter, such that as closer its value to zero, the more similar the two spanners are in this respect.
For each point $p \in V$, we define $C(p)$ as the set of cones which have p as their apex. Let $c \in C(p)$, if a point q lies in c, then it can be shown that there is an evident edge for the point pair (p, q) in the current spanner (Lemma 2). Due to this, at the moment when (p, q) is to be processed, the proposed algorithms at first examine existence of such cone $c \in C(p)$. We show that the cost of this examination is constant (Lemma 3).

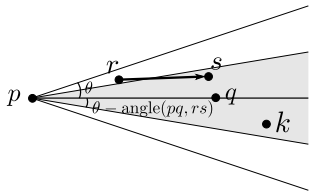

Figure 1: Illustration of Lemma 3. (r, s) is an α-evident edge for (p, q) and an evident edge for (p, k).

Cones are added during the construction of the α-Gap greedy spanner. When a point pair (p, q) is to be processed, if a cone $c \in C(p)$ cannot be found that contains q, search for finding an α-evident edge for (p, q) will be performed and then a cone is added to $C(p)$. We will show that for each point $p \in V$, the number of times this search is performed is a constant that depends only on α (Lemma 4).
Let c be a cone with $\overrightarrow{pq}$ as bisector that is added to $C(p)$ after α-evident edge search performed for a pair (p, q). The angle of c depends on the result of α-evident edge search. If an α-evident edge (r, s) is found for the pair (p, q), the angle of c is set to $2(\theta - \text{angle}(pq, rs))$. Note that in this case, we have $|pr| \leq w|rs|$, otherwise the cone c is not added to $C(p)$. Lemma 3 shows that the pair (r, s) is an evident edge for all pairs (p, k), where $k \in V$ is a point that lies in c (See Figure 1). It's easy to see that the smaller the angle between (r, s) and (p, q), the larger angle of c is. Note that α parameter limits the minimum size of c. If no α-evident edge is found for the pair (p, q), the angle of c is set to 2θ.
Lemma 2 Let θ and α be real numbers such that $0 < \theta < \frac{\pi}{3}$ and $0 < \alpha < \theta$. Suppose (r, s) is an α-evident edge for (p, q), and k is a point on $\mathbb{R}^d$ such that:

1. $|pq| \leq |pk|$,
2. $\text{angle}(pk, pq) \leq \theta - \text{angle}(pq, rs)$.

Then (r, s) is an evident edge of (p, k).

Proof. It is easy to see that $|rs| \leq |pk|$, so if we show $\text{angle}(pk, rs) \leq \theta$, then based on Lemma 2, it can be concluded that (r, s) is an evident edge for (p, k).
To show that $\text{angle}(pk, rs) \leq \theta$, we use property (2) of Lemma 3:

$$\begin{aligned} &\text{angle}(pk, rs) \\ &\qquad \leq \text{angle}(pk, pq) + \text{angle}(pq, rs) \\ &\qquad \leq \theta - \text{angle}(pq, rs) + \text{angle}(pq, rs) \\ &\qquad = \theta. \end{aligned}$$

The lemma follows. □

3.1 First algorithm

Our first algorithm is a simple extension of GapGreedy algorithm. It starts with an empty graph $G = (V, \emptyset)$ and processes the input pairs of points in a non-decreasing order of their distances. At the step when a pair (p, q) is to be processed, it acts as mentioned in the previous section, i.e., it examines if there exists a cone $c \in C(p)$ that contains q. If such a cone c is not found, α-evident edge search operation is performed for (p, q). In Algorithm 1, the pseudo-code of the proposed algorithm is given.

Algorithm 1: α-GapGreedy(V, w, θ, α)

Input : A set V of n points in the plane and three real numbers $0 \leq w < 1 - 2\sin(\frac{(\sin\theta/2)}{2})$, $0 < \theta < \frac{\pi}{3}$ and $0 < \alpha < \theta$

Output: An α-Gap greedy spanner for V

```
1  Sort the 2*C(n,2) pairs of distinct points in non-decreasing order of their distances and
   store them in list L;
2  E ← ∅;
3  C(p) ← ∅, ∀p ∈ V;
4  G = (V, E);
5  foreach (p, q) ∈ L do
6      if q ∈ C(p) then
7          continue
8      end
9      Let e = (r, s) be an α-evident edge found for (p, q);
10     if e is null then
11         E ← E ∪ {(p, q)};
12         c_p(2θ, q) ← cone of angle 2θ with apex at p and bisector pq→;
13         C(p) ← C(p) ∪ c_p;
14     end
15     else
16         if |pr| ≤ w|rs| then
17             γ ← θ − angle(pq, rs);
18             c_p(2γ, q) ← cone of angle 2γ with apex at p and bisector pq→;
19             C(p) ← C(p) ∪ c_p;
20         end
21     end
22 end
23 return G = (V, E)
```

3.1.1 Algorithm Analysis

Lemma 3 The number of cones stored by each point is at most $\frac{2\pi}{\alpha}$.

Proof. Suppose (p, q) and (p, r) are two pairs of points that α-evident edge search have been performed for, and their corresponding cones have been added to $C(p)$. Without loss of generality, we assume that (p, q) is processed before (p, r). According to Algorithm 1, $\angle rpq$ is at least α, because otherwise, r lies in a cone with $\overrightarrow{pq}$ as bisector and thus, according to Lemma 3, there is a t-path between p and r. In other words, angle$(pq, pr) > \alpha$, so we can easily see that $|C(p) < \frac{2\pi}{\alpha}$. □

Lemma 4 The number of times that α-evident edge search is performed by the proposed algorithm is at most $2\frac{2\pi}{\alpha}n$.

Proof. Assume that (r, s) is α-evident edge that is found for (p, q). According to the definition of the α-evident edge, we have $|pr| \leq w|rs|$ or $|qs| \leq w|rs|$. If we have $|pr| \leq w|rs|$, then the algorithm adds a cone to $C(p)$ after searching for (p, q). In case that $|qs| \leq w|rs|$, after searching for an α-evident edge for (q, p), the algorithm adds a cone to $C(q)$. In other words, when α-evident edge search is performed for (p, q) or (q, p), at least one cone is added to the set of cones of p or q, respectively. On the other hand, according to Lemma 3.1.1, the number of cones that is maintained by each point is at most $\frac{2\pi}{\alpha}$. Since at most two α-evident edge searches are performed for every cone, the desired result is obtained. □

Lemma 5 Let θ, w and α be real numbers such that $0 < \theta < \frac{\pi}{3}$ and $0 \leq w < 1 - 2\sin(\frac{(\sin\theta/2)}{2})$, $0 < \alpha < \theta$, and let V be a set of n points in the plane.

1. Algorithm α-GapGreedy(V, w, θ, α) computes a graph in which each vertex has degree at most $2\lceil\frac{2\pi}{\theta-\alpha}\rceil$.

2. If $w > 0$, then the weight of this graph is less than $\lceil\frac{2\pi}{\theta-\alpha}\rceil(1 + \frac{2}{w})\log n$ times the weight of a minimum spanning tree of V.

Proof. The proof is similar to Lemma 7.2.2 of Narasimhan and Smid [13], we only need to substitute $\theta - \alpha$ for θ. □

Theorem 6 α-GapGreedy(V, w,θ, α) computes the α-Gap greedy spanner in $O(n^2 \log n + \frac{n^2}{\alpha})$ time and $O(n^2 + \frac{n}{\alpha})$ space.

Proof. The algorithm starts by sorting $2\binom{n}{2}$ of input pairs of points that requires $O(n^2 \log n)$ time. According to Lemma 3.1.1, the number of times the algorithm searches for an α-evident edge is $O(\frac{n}{\alpha})$. Every such search costs $O(n)$, therefore processing of the pairs has the cost of $O(\frac{n^2}{\alpha})$, thus total time cost of the algorithm is $O(n^2 \log n + \frac{n^2}{\alpha})$. The list of points pairs requires $O(n^2)$ space and space cost of the cones based on Lemma 3.1.1 is at most $\frac{\pi}{\alpha}n$, so space cost of the algorithm will be $O(n^2 + \frac{n}{\alpha})$. □

3.2 Second Algorithm

In our second algorithm, named LinearSpace-α-GapGreedy, we were inspired by the method of Bouts *et al.* [7], used to construct *Path-greedy* spanner, to reduce memory usage of the first proposed algorithm. In this method, the set of input pairs of points are partitioned into $O(n)$ sets, and thus processing of $\binom{n}{2}$ pairs of points is convert to processing of $O(n)$ sets of pairs of points.
Let S_p be a set of pair of points that have p as starting point, e.g. $S_p = \{(p, q) \mid q \in V\}$. We refer to p as the center of S_p. We denote the *candidate* of S_p as $C(S_p)$ and define it as a pair (p, q) such that (i) there is no α-evident edge for (p, q) in the current spanner, and (ii) q be the closest point to p. We say that p is *dirty* if the newly added edge to the spanner is an α-evident edge for $C(S_p)$.
The LinearSpace-α-GapGreedy algorithm considers S_p for each point $p \in V$ and repeatedly selects the smallest $C(S_p)$. Suppose (p, q) is the current smallest candidate. If p is not dirty, (p, q) will be added to the spanner as a new edge, otherwise, the new candidate for S_p is identified and replaced using an operation named *ClosestPair*. When an edge (p, q) is added to the spanner, a cone is also added to $C(p)$, as described in the previous section. The pseudo-code of LinearSpace-α-GapGreedy is shown in Algorithm 2.
When *ClosestPair* operation is called for S_p, unprocessed pairs of S_p are examined in a non-decreasing order of their distances and the first found pair without an α-evident edge is returned as output. When *ClosestPair* operation processes a pair (p, q), at first examines $C(p)$ to find a cone containing q. If no such cone is found, α-evident edge search is performed. Suppose (r, s) is the α-evident edge found, if we have $|pr| \leq w|rs|$, a cone of angle $2(\theta - \text{angle}(pq, rs))$ with $\overrightarrow{pq}$ as its bisector is added to $C(p)$. The *ClosestPair* operation is presented in Algorithm 3.
According to Algorithm 3, *ClosestPair* operation is completed in two cases: 1) Q_p is empty, and 2) the extracted element from Q_p has no α-evident edge. The first case occurs when all elements of S_p are processed and the desired element is not found, so it is the last time the operation is called for S_p. The second case occurs when α-evident edge search operation is performed and the current element is identified as a new candidate of S_p. In fact, except for the last time that this operation is called for S_p, at the other times, the completion requires at least one call to α-evident edge search operation.
Observation 1 α-evident edge search operation is called at least once every time *ClosestPair* operation is called for a set with the center point p (except for the last time).
Lemma 7 The number of times that *ClosestPair* operation is called is at most $2\frac{2\pi}{\alpha}n$.

Proof. According to Lemma 3.1.1, the number of times the α-evident edge search is called is at most $2\frac{2\pi}{\alpha}n$. On the other hand, according to Observation 3.2, every time *ClosestPair* operation is called for a set (except for the last time), α-evident search operation is also called at least once. So total number of times that *ClosestPair* operation is called is at most $2\frac{2\pi}{\alpha}n$. □

Theorem 8 Algorithm LinearSpace-α-GapGreedy(V, w, θ, α) computes the α-Gap greedy spanner in $O(\frac{n^2}{\alpha} + n^2 \log n)$ time and $O(\frac{n}{\alpha})$ space.

Algorithm 2: LinearSpace-α-GapGreedy(V, w, θ, α)

Input : A set V of n points in the plane and three real numbers $0 \leq w < 1 - 2\sin(\frac{(\sin\theta/2)}{2})$, $0 < \theta < \frac{\pi}{3}$ and $0 < \alpha < \theta$

Output: An α-Gap greedy spanner for V

```
1  E ← ∅;
2  C(p) ← ∅, ∀p ∈ V;
3  Q ← {(a, b) | b is the nearest point to a};
4  G = (V, E);
5  foreach a ∈ V do
6  |  Mark a as clean;
7  end
8  while Q ≠ ∅ do
9  |  (p, q) ← ExtractMin(Q);
10 |  if p is dirty then
11 |  |  k ← ClosestPair(C(p), p);
12 |  |  if k is not null then
13 |  |  |  Insert (p, k) into Q;
14 |  |  end
15 |  end
16 |  else
17 |  |  E ← E ∪ {(p, q)};
18 |  |  Insert (p, q) into Q;
19 |  |  c_p(2θ, q) ← cone of angle 2θ with apex at p and bisector pq→;
20 |  |  C(p) ← C(p) ∪ c_p;
21 |  |  foreach (a, b) ∈ Q do
22 |  |  |  if (p, q) is an α-evident edge for (a, b) then
23 |  |  |  |  Mark a as dirty;
24 |  |  |  end
25 |  |  end
26 |  end
27 end
28 return G = (V, E)
```

Proof. The only information held by the algorithm is the cones, so according to Lemma 3.1.1, the space cost of the algorithm is $O(\frac{n}{\alpha})$.
Determining the nearest point to the central point of each set, that is done at the beginning of the algorithm, costs $O(n^2)$. According to Lemma 3.2, *ClosestPair* operation is called at most $2\frac{2\pi}{\alpha}n$. Each time this operation is performed, at most n elements are inserted in queue. This step can be done by a specific implementation at a cost of $O(n)$, so the total cost of this step is $O(\frac{n^2}{\alpha})$. For a specific set, a point is extracted from the queue only once, so the overall cost of this extraction for each set is $O(n \log n)$, and therefore total

Algorithm 3: ClosestPair$(C(p), p)$

Input : A set $C(p)$ of cones and a point p
Output: New candidate for the S_p (or *null* if there is no candidate)

```
1  Let (p, k) be the current candidate of S_p;
2  Q_p ← ∅;
3  foreach a ∈ V do
4  |  if |pa| > |pk| then
5  |  |  Insert a into Q_p;
6  |  end
7  end
8  while Q_p ≠ ∅ do
9  |  q ← ExtractMin(Q_p);
10 |  if q ∈ C(p) then
11 |  |  continue
12 |  end
13 |  Let e = (r, s) be an α-evident edge found for (p, q);
14 |  if e is null then
15 |  |  return q
16 |  end
17 |  if |pr| ≤ w|rs| then
18 |  |  γ ← θ − angle(pq, rs);
19 |  |  c_p(2γ, q) ← cone of angle 2γ with apex at p and bisector pq→;
20 |  |  C(p) ← C(p) ∪ c_p;
21 |  end
22 end
23 return null
```

cost of this extraction is $O(n^2 \log n)$.
After extracting a pair (p, q), we first examine whether q lies in a cone of p. Note that the number of cones held by each point is $O(\frac{1}{\alpha})$ (Lemma 3.1.1), and this check is performed exactly once for each pair, so the total cost of this step is $O(\frac{n^2}{\alpha})$. According to Lemma 3.1.1, the number of times that α-evident search operation is called is $O(\frac{n}{\alpha})$, therefore, the cost imposed by this operation in the algorithm is $O(\frac{n^2}{\alpha})$. After adding a new edge to the spanner, at most n candidates must be examined. The cost of this examination is $O(n)$, and since the number of edges added to the spanner is $O(n)$, the total cost of this step is $O(n^2)$, thus total cost of the algorithm will be $O(\frac{n^2}{\alpha} + n^2 \log n)$.
□

4 Construction of the α-Gap greedy Spanner on Random Point Set in Unit Square

Suppose n points are randomly and uniformly distributed in a unit square. In this section, we show that by making some changes to our first proposed algorithm, it is possible to create the α-Gap greedy spanner on such points in expected $O(n \log n)$ time. The method used in this section is inspired by work of Bar-on and Carmi [5], which was proposed to construct the Path-greedy spanner on such points. To show that the proposed changes lead to the expected running time of $O(n \log n)$ for constructing the α-Gap greedy spanner, it is necessary to show the followings:

1. For each point, α-evident edge search is called a constant number of times, which follows from Lemma 3.1.1,

2. The expected number of edges examined in α-evident edge search is constant.

Note as the points are randomly distributed in unit square, the expected number of points at the distance of at most x from p is $\Theta(x^2 n)$. Also, based on Lemma 3.1.1, the number of α-evident edge searches performed by the algorithm for a point is $O(\frac{1}{\alpha})$. Each of these searches define a cone with apex at p of angle at least α, such that no α-evident edge search from p is performed to any point in this cone. we set $y = \frac{1}{\alpha}$ and $x = \frac{y}{\sqrt{n}}$. In this case, the expected number of points around each point in a distance x will be $\Theta(y^2) = \Theta(\frac{1}{\alpha^2})$.
We divide the plane into y cones with apex at p of equal angles. It's easy to see that the probability of having a cone that does not contain a point from the set of points in a distance x from p is at most $y(1 - \frac{1}{y})^{y^2}$. Let Q be a set consists of all the points q such that α-evident edge search has performed for (p, q). Consider $q \in Q$ be the farthest point of Q from p, so the expected Euclidean distance between p and q is at most x. In other words, with high probability, the farthest pair of points that search for an α-evident edge will be performed, has distance at most x.
To show the expected number of edges examined in α-evident edge search is constant, we use the locality of evidence property. As in Lemma 2 it was shown, the (α-) evident edges of a pair (p, q) are placed in a distance at most $|pq|$ from p and q. On the other hand, the number of edges of the spanner at such a distance is limited. The expected number of points in the distance $|pq|$ from point p and q are:

$$\begin{aligned} 2(|pq|^2 n) & \\ & \leq 2x^2 n \\ & = 2y^2. \end{aligned}$$

According to Lemma 3.1.1, the degree of vertices in the α-Gap greedy spanner is at most $2\lceil \frac{2\pi}{\theta - \alpha} \rceil$. With some simplifications, the expected number of edges of the spanner that should be examined from one point in α-evident edge search is $O(\frac{1}{\alpha^2(\theta - \alpha)})$.

Note that the main cost of the first proposed algorithm is implied from sorting all pairs of points. It is possible to obtain the next pair in the sorted order at an expected cost of $O(\log n)$ without sorting all the pairs. The method is as follows [5]. The unit square is divided to $n \times n$ grid cells with side length $\frac{1}{n}$. The points in every non-empty grid cell is mapped into a hash table of size $3n$.
For each point $p \in V$, an initially empty minimum heap H_p is maintained, which contains a subset of the pairs that include p. In addition, the top element of each H_p is maintained in a minimum heap H. The algorithm uses the hash table to scan all the cells of distance at most x from every point $p \in V$ to find all the points in these cells. The points found are added to H_p according to their Euclidean distance from p.
The pairs are held in the main heap H in an increasing order. Assuming the extracted pair from H is belong to a point p, the next pair added to H is the minimum pair in H_p. Note that if it needed, the distance to the scanned cells is increased to ensure the correctness of the heaps, but the only cells that are not contained in cones of $C(p)$ are scanned to add more pairs to H_p. Therefore, the total expected running time of the algorithm is $O(\frac{1}{\alpha^2(\theta-\alpha)} n \log n)$.

5 Experimental Results

In this section, we compare the α-Gap greedy construction algorithms experimentally. The experiments were performed on sets consisting of 250 to 4000 points with uniform and clustered distributions. We followed the method proposed by Alewijnse *et al.* [1, 2] to generate points with clustered distribution. In the experiments, four different configurations of $C1$ - $C4$ were considered. The specifications of each of these configurations are shown in Table 1.

Table 1: Specification of configuration used in the experiments

Parameter	Stretch factor=2		Stretch factor=1.2	
	$C1$	$C2$	$C3$	$C4$
w	0.1555	0.2045	0.0655	0.008870
θ	0.174533	0.087267	0.034907	0.139626

The implementation of the algorithms was done using C++ language, and compiled into machine codes using Visual Studio 2015 compiler. The experiments were performed on a machine with an Intel Core i5-2410M processor with 4GB of main memory on the Windows 7 operating system.

5.1 Construction Time

In this section, we investigate the construction time of the α-Gap greedy spanner that is imposed by the two proposed algorithms. The results are divided into two parts. Section 5.1.1 examines the effect of α-evident edge *selection policy* used by the proposed

Figure 2: Construction time of the α-Gap greedy spanner imposed by the proposed algorithms with different α-evident edge selection policies. Configuration $C1$ (a and b), Configuration $C2$ (c and d)

Figure 3: Construction time of the α-Gap greedy spanner imposed by the proposed algorithms with different α-evident edge selection policies. Configuration $C3$ (a and b), Configuration $C4$ (c and d)

algorithms on the α-Gap greedy spanner construction speed, and in section 5.1.2, we examine the effect of α value on this property.

5.1.1 Effect of the α-evident edge selection policies on the construction time

For a pair of points, there may be more than one α-evident edge in a spanner, so the proposed algorithms may have more than one choice at the time of searching for an α-evident edge. In order to determine the effectiveness of possible choices during the construction of the α-Gap greedy spanner, two implementations have been done for each of the two proposed algorithms. In the first implementation, the first eligible edge found, is considered as the α-evident edge. In the second one, we use the best eligible edge as the α-evident edge. The best eligible edge is the edge whose cone has bigger angle than the other candidates. Note that the output spanner produced by these two implementations is the same.
The Figures 2 and 3 present the results of the first proposed algorithm with first candidate policy ($AGFC$), the first proposed algorithm with best candidate policy ($AGBC$), the second proposed algorithm with first candidate policy ($LSFC$) and the second proposed algorithm with best candidate policy ($LSBC$). In the results shown, we use θ^3 for the α parameter.
As can be seen in almost all experiments, the algorithms with a first candidate policy spend less time for constructing the α-Gap greedy spanner, and this advantage increases with increasing number of points (especially in clustered distribution). The results also show that in uniform distribution, the range of variations in construction time of the α-Gap greedy spanner is smaller than that of cluster distribution.
According to the results, in subsequent experiments, the policy of first candidate has been used for the proposed algorithms.

5.1.2 Effect of α parameter on the construction time

In this section, we examine the effect of α parameter value on the construction speed of the α-Gap greedy spanner. The values used for this parameter are equal to $\theta^{1.5}$, θ^2 and θ^3. For the comparison purposes, the Gap-greedy spanner is also constructed using

Figure 4: Comaprison of the proposed α-Gap greedy spanner construction algorithms and *GapGreedy* algorithm. Configuration $C1$ (a and b), Configuration $C2$ (c and d)

Figure 5: Comaprison of the proposed α-Gap greedy spanner construction algorithms and *GapGreedy* algorithm. Configuration $C3$ (a and b), Configuration $C4$ (c and d)

the *GapGreedy* algorithm (*GapGR*) on the set of input points and the construction time is shown in the following results along the results related to the proposed algorithms (Figures 4,5).
As can be seen, the superiority of the proposed algorithms over the *GapGreedy* algorithm is obvious, however, in the clustered distribution of points, the degree of superiority is slightly higher than in the case where the distribution of points is uniform. Based on the time cost analysis of the algorithm, it is obvious that as the number of points increases, the difference between the proposed algorithms increases compared to the *GapGreedy* algorithm.
Another observation in the displayed results is that the value of α parameter in the uniform distribution has a greater effect on the construction time of the proposed algorithms, while in the clustered distribution, this effect is negligible. Finally, as you can see, as the closer the value of α parameter is to zero, the shorter the construction time of the α-Gap greedy spanner due to the smaller number of edges (see the results in Section 5.2).

5.2 Quality of the α-Gap greedy spanner

In this section, the results related to qualitative features of the α-gap greedy spanner are given. The features include weight, maximum degree of vertices and size of the longest edge in the spanner. In order to investigate the effect of the α parameter, the α-Gap greedy spanner was constructed for different values of this parameter, as Section 5.1.2. We also include the above qualitative features associated with the Gap-greedy spanner built on the same set of input points so that we can examine the differences between the resulting spanners. Note that regarding the weight of graph and the heaviest edge, the ratio of their value in the α-Gap greedy to that of the Gap-greedy spanner is shown.
As expected, as α value decreases and approaches 0, the maximum degree of vertices and the weight of the α-Gap greedy spanner get closer to the values associated with these characteristics in the Gap-greedy spanner. Note that this observation is the same in all experiments performed (different number of points and distributions, as well as different values for θ and w parameters).
Also note that the quality of the resulting α-Gap greedy spanner is affected more by changing the value of α parameter when the distribution of points is uniform. As shown in Tables 2-3, changing this parameter, if the distribution of points is uniform, has a greater

Table 2: Unifrom distribution

Configuration	α	Number of Points=1000				Number of Points=4000			
		% Weight	Maximum Degree		% Edge Size	% Weight	Maximum Degree		% Edge Size
			$\alpha-$GG	GG			$\alpha-$GG	GG	
$C1$	0.00532	1.0307	52	50	0.7999	1.0302	54	54	1.1536
	0.03046	1.2211	60	50	0.9936	1.2208	64	54	1.2374
	0.07291	1.7416	83	50	0.9573	1.7331	87	54	1.1098
$C2$	0.00066	1.0056	88	86	0.9882	1.0058	94	94	0.9452
	0.00762	1.0688	94	86	1.1801	1.0837	100	94	1.1147
	0.02578	1.3417	117	86	1.0136	1.3547	121	94	1.0626
$C3$	0.00004	1.0018	212	212	1.0000	1.0013	236	236	1.0000
	0.00122	1.0348	220	212	1.0000	1.0393	242	236	1.1002
	0.00652	1.2305	255	212	1.0020	1.2317	282	236	1.1144
$C4$	0.00272	1.0228	66	64	0.9873	1.0350	70	70	1.0082
	0.01950	1.1981	76	64	1.0616	1.2091	80	70	1.0004
	0.05217	1.7223	98	64	1.0749	1.7867	107	70	1.0377

Table 3: Clustered distribution

Configuration	α	Number of Points=1000				Number of Points=4000			
		% Weight	Maximum Degree		% Edge Size	% Weight	Maximum Degree		% Edge Size
			$\alpha-$GG	GG			$\alpha-$GG	GG	
$C1$	0.00532	1.0210	34	34	1.0000	1.0321	46	46	1.0709
	0.03046	1.1237	38	34	0.9919	1.2308	50	46	1.2684
	0.07291	1.5053	44	34	0.9887	1.6962	64	46	1.2769
$C2$	0.00066	1.0071	44	44	1.0000	1.0012	68	68	1.0000
	0.00762	1.0650	46	44	0.9919	1.0520	68	68	1.1308
	0.02578	1.1935	50	44	0.9887	1.2878	78	68	1.1308
$C3$	0.00004	1.0000	64	62	1.0000	0.9992	100	100	1.0000
	0.00122	1.0650	65	62	1.0000	1.0146	102	100	0.9990
	0.00652	1.1935	70	62	1.0735	1.1115	112	100	1.0150
$C4$	0.00272	0.9959	44	44	0.9822	1.0096	56	56	1.1119
	0.01950	1.0597	46	44	1.0340	1.1195	58	56	1.0696
	0.05217	1.2794	48	44	0.9451	1.3547	76	56	1.0584

impact on the characteristics of the resulting the α-Gap greedy spanner (maximum degree of vertices and weight). This observation justifies the results of the previous section, in which the construction time by the proposed algorithms for points with clustered distributions at different α values did not make a significant difference.
Finally, the above observation is not true about the size of the largest edge. In fact, as can be seen, changing α value does not have a significant effect on this feature in the α-Gap greedy spanner compared to the Gap-greedy spanner.

References

[1] Alewijnse, S. P., Bouts, Q. W., Alex, P., and Buchin, K. Computing the greedy spanner in linear space. *Algorithmica 73*, 3 (2015), 589–606.

[2] Alewijnse, S. P., Bouts, Q. W., Alex, P., and Buchin, K. Distribution-sensitive construction of the greedy spanner. *Algorithmica 78*, 1 (2017), 209–231.

[3] Arya, S., and Smid, M. Efficient construction of a bounded-degree spanner with low weight. *Algorithmica 17*, 1 (1997), 33–54.

[4] Bakhshesh, D., and Farshi, M. Improving space and time complexity of the gap-greedy spanner algorithm. *The CSI Journal on Computer Science and Engineering 14*, 1 (2016), 6–18.

[5] Bar-On, G., and Carmi, P. δ-greedy t-spanner. In *Workshop on Algorithms and Data Structures* (2017), Springer, pp. 85–96.

[6] Bose, P., Carmi, P., Farshi, M., Maheshwari, A., and Smid, M. Computing the greedy spanner in near-quadratic time. *Algorithmica 58*, 3 (2010), 711–729.

[7] Bouts, Q. W., ten Brink, A. P., and Buchin, K. A framework for computing the greedy spanner. In *Proceedings of the thirtieth annual symposium on Computational geometry* (2014), ACM, p. 11.

[8] Carmi, P., and Smid, M. An optimal algorithm for computing angle-constrained spanners. *Journal of Computational Geometry 3*, 1 (2012).

[9] Das, G., and Narasimhan, G. A fast algorithm for constructing sparse euclidean spanners. *International Journal of Computational Geometry & Applications 7*, 04 (1997), 297–315.

[10] Farshi, M., and Gudmundsson, J. Experimental study of geometric t-spanners. *Journal of Experimental Algorithmics (JEA) 14* (2010), 1–3.

[11] Gudmundsson, J., Levcopoulos, C., and Narasimhan, G. Improved greedy algorithms for constructing sparse geometric spanners. In *Scandinavian Workshop on Algorithm Theory* (2000), Springer, pp. 314–328.

[12] Li, X.-Y. *Wireless ad hoc and sensor networks: theory and applications.* Cambridge University Press, 2008.

[13] Narasimhan, G., and Smid, M. *Geometric spanner networks.* Cambridge University Press, 2007.

[14] Soares, J. Approximating euclidean distances by small degree graphs. *Discrete & Computational Geometry 11*, 2 (1994), 213–233.