A framework for computing the greedy spanner

Citation for published version (APA):

DOI:
10.1145/2582112.2582154

Document status and date:
Published: 01/01/2014

Document Version:
Accepted manuscript including changes made at the peer-review stage

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher’s website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
A Framework for Computing the Greedy Spanner

Quirijn W. Bouts  Alex P. ten Brink  Kevin Buchin
Department of Mathematics and Computer Science
TU Eindhoven, Eindhoven, The Netherlands
q.w.bouts@tue.nl  k.a.buchin@tue.nl

ABSTRACT
The highest quality geometric spanner (e.g., in terms of edge count, both in theory and in practice) known to be computable in polynomial time is the greedy spanner. The state-of-the-art in computing this spanner are a $O(n^2 \log n)$ time, $O(n^2)$ space algorithm and a $O(n^2 \log^2 n)$ time, $O(n)$ space algorithm, as well as the ‘improved greedy’ algorithm, taking $O(n^2 \log n)$ time in the worst case and $O(n^2)$ space but being faster in practice thanks to a caching strategy.

We identify why this caching strategy gives speedups in practice. We formalize this into a framework and give a general efficiency lemma. From this we obtain many new time bounds, both on old algorithms and on new algorithms we introduce in this paper. Interestingly, our bounds are in terms of the well-separated pair decomposition, a data structure not actually computed by the caching algorithms.

Specifically, we show that the ‘improved greedy’ algorithm has a $O(n^2 \log n \log \Phi)$ running time (where $\Phi$ is the spread of the point set) and a variation has a $O(n^2 \log^2 n)$ running time. We give a variation of the linear space state-of-the-art algorithm and an entirely new algorithm with a $O(n^2 \log n \log \Phi)$ running time, both of which improve its space usage by a factor $O(1/(t − 1))$, where $t$ is the dilation of the spanner.

We present experimental results comparing all the above algorithms. The experiments show that our new algorithm is much more space efficient than the existing linear space algorithm - up to 200 times when using low $t$ - while being comparable in running time and much easier to implement.

Categories and Subject Descriptors
F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—geometrical problems and computations

General Terms
Algorithms, Theory

Keywords
computational geometry, spanners, well-separated pair decomposition

1. INTRODUCTION
We consider graphs on sets of points in $\mathbb{R}^d$ whose edges are weighted according to the Euclidean distance measure. If the shortest path in the graph between two points is at most $t$ times larger than the direct Euclidean distance, then we say that these points have a $t$-path; if all pairs of points have $t$-paths, the graph is a $t$-spanner.

For any $t > 1$, we can efficiently find a $t$-spanner with $O\left(\frac{n}{(t-1)^{d-1}}\right)$ edges for fixed $d$ [10]. These graphs approximate Euclidean distances while containing only few edges, making them a useful tool in many areas.

For example, in wireless network design, bounded degree spanners are used to minimize problems with interference while maintaining connectivity [7]. Since their introduction in network design [11] and geometry [5], much research has been done on spanners (see [8, 10] for surveys). They have been used as a tool in various geometric and distributed algorithms since.

Some well-known graphs are $t$-spanners for fixed $t$, such as the Delaunay triangulation. There also exist many constructions parameterizable with arbitrary $t > 1$ which produce a $t$-spanner for that value, each having an advantage over the other construction methods. Various quality measures of spanners are: total number of edges, maximum degree, total weight, computation speed and space usage, scaling to higher dimensions and diameter. For an in-depth treatise of many spanner constructions, see the book [10].

The spanner construction resulting in the highest quality spanners known to be computable in polynomial time is the greedy spanner [9]. This spanner has maximum degree $O\left(\frac{1}{(t-1)^d}\right)$ and total weight $O\left(\frac{w(MST)}{(t-1)^d}\right)$, where $w(MST)$ is the weight of a minimal spanning tree on the same input [10]. Also in practice its weight and degree is much smaller than that of other well-known algorithms. Figure 1 shows the difference between the greedy spanner and the $\Theta$-spanner.

After proving the above qualities of the greedy spanner, the initial $\Theta(n^2 \log n)$ time, $\Theta(n^2)$ space algorithm was improved to a $\Theta(n^2 \log n)$ time, $\Theta(n^2)$ space algorithm [2], a $O(n^2 \log^2 n)$ time, $O\left(\frac{n}{(t-1)^d}\right)$ space algorithm [1] and a
that the state-of-the-art sub-cubic-time algorithms are near-
framework generalizes the (ad-hoc) arguments used to prove
ation on that algorithm for which no cubic worst-cases are
algorithm [6] (which also uses a caching strategy) and a vari-
framework to obtain time bounds for the Improved-Greedy
algorithms really makes the algorithm efficient. We present
these time bounds, we prove that the caching used by our

Figure 1: The Greedy-spanner (left) and Θ-graph
(right) on the USA, zoomed in on Florida, using
t = 2 and for the Θ-graph, k = 6 for which it was
recently proven it also achieves a dilation of 2.

\[ O(n^3 \log n) \] time, \[ O(n^2) \] space algorithm that works well in
practice [6] (time bounds assume fixed t). While the linear
space algorithm allows us to compute greedy spanners on
much larger sets than before – quadratic space usage limits
us to about 13,000 points – the space usage of the algorithm
is still quite large (especially for low t) and the algorithm
requires extensive tweaking to have acceptable performance.

We give two new algorithms for the greedy spanner: a
variation of the known linear space algorithm and an enti-
tirely new algorithm. Both use \( O\left(\frac{n}{(t-1)\log t}\right) \) space, giving
a \( O\left(\frac{1}{t}\right) \) improvement over the state-of-the-art in theory.
In practice, the entirely new algorithm uses up to a factor
200 less space than the other two algorithms, while being
comparable in speed for high values of t (factor 2 slower for
uniformly distributed points with \( t = 2 \)) and being faster
for lower values of t (about a factor 9 for \( t = 1.1 \) on the
same data), while being by far the simplest to implement.
The variation improves both the speed and space usage of
the original for lower values of t significantly, while staying
comparable for higher t.

However, the time usage of both algorithms is hard to an-
alyze. We prove a \( O(n^2 \log^2 n) \) time bound for the variation
and a \( O(n^2 \log n \log \Phi) \) bound (where \( \Phi = \frac{\max_{u,v \in V} |uv|}{\min_{u,v \in V} |uv|} \)
is the spread of the point set) for the new algorithm. To get
these time bounds, we prove that the caching used by our
algorithms really makes the algorithm efficient. We present
a generalized argument in this paper in the form of a frame-
work for analyzing greedy spanner algorithms. We use this
framework to obtain time bounds for the Improved-Greedy
algorithm [6] (which also uses a caching strategy) and a vari-
ation on that algorithm for which no cubic worst-cases are
known, explaining its performance in practice and provid-
ing upper bounds that match known lower bounds [2]. The
framework generalizes the (ad-hoc) arguments used to prove
that the state-of-the-art sub-cubic-time algorithms are near-

The framework and the arguments used in the applica-
tion of the framework use the well-separated pair decom-
position [3] extensively. This data structure was already
used in the linear space algorithm and has several known
connections with the greedy spanner [2]. Our framework
adds another connection to this: the time bound we prove
on our algorithm is \( O(n \log n \sum_{i=1}^{m} |A_i| + |B_i|) \) (which is
\( O(n^2 \log n \log \Phi) \)), where \( m \) is the number of well-separated
pairs and \( \{A_i, B_i\} \) is the \( i \)-th well-separated pair and \( A_i \)
and \( B_i \) are sets of points. The algorithm itself however does
not compute this well-separated pair decomposition, which
partly explains its low space usage and ease of implemen-
tation, as the WSPD provides neither.

The rest of the paper is organized as follows. In Section 2
we review a number of well-known definitions, algorithms
and results. In Section 3 we give our new algorithm. In
Section 4 we present our framework, give our efficiency lemma
and give a detailed example application. In Section 5 we
list the results that follow from our framework. Finally, in
Section 6 we present experimental results for all relevant
algorithms.

2. PRELIMINARIES

Let \( V \) be a set of points in \( \mathbb{R}^d \), and let \( t \in \mathbb{R} \) be the
intended dilation (\( 1 < t \)). Let \( G = (V, E) \) be a graph on \( V \).
For two points \( u, v \in V \), we denote the Euclidean distance
between \( u \) and \( v \) by \(|uv|\), and the distance in \( G \) by \( \delta_G(u, v) \)
(if the graph \( G \) is clear from the context we will simply write
\( \delta(u, v) \)). We say a pair of points has a \( t \)-path if \( \delta(u, v) \leq t \cdot |uv| \).
A graph \( G \) has dilation \( t \) if all pairs of points have
\( t \)-paths. In this case we say that \( G \) is a \( t \)-spanner.

Algorithm GreedySpannerOriginal(V, t)
1. \( E \leftarrow \emptyset \)
2. for every pair of distinct points \( (u, v) \) in ascending order
   of \(|uv|\)
3. \hspace{1cm} do if \( \delta_G((u, v)) > t \cdot |uv| \)
   \hspace{1cm} then add \((u, v)\) to \( E \)
4. return \( E \)

Consider algorithm GreedySpannerOriginal introduced by
Keil [9]. Given \( V \) and \( t \), we define the greedy spanner as the
result of this algorithm with parameters \( V \) and \( t \). Obvi-
ously, the greedy spanner is a \( t \)-spanner. We also have the
following [10, Corollary 15.1.3]:

**Lemma 2.1.** The greedy spanner has at most \( O\left(\frac{n}{t-\frac{1}{\log t}}\right) \)
edges.

We will now treat the well-separated pair decomposition
(WSPD) as introduced by Callahan and Kosaraju [3, 4]. A
WSPD is parameterized with a separation constant \( s \in \mathbb{R} \)
with \( s > 0 \). It is a set of pairs of nonempty subsets of \( V \).
Let \( m \) be the number of pairs in a WSPD. We number the
pairs, and denote every pair as \( \{A_i, B_i\} \) with \( 1 \leq i \leq m \).
If \( u \) and \( v \) are distinct points, then we say that \((u, v)\) is \( \in \)
a well-separated pair \( \{A_i, B_i\} \) if \( u \in A_i \) and \( v \in B_i \) or \( v \in A_i \)
and \( u \in B_i \). A WSPD has the property that for every pair
of distinct points \( u \) and \( v \), there is exactly one \( i \) such that
\((u, v) \) is \( \in \) \( \{A_i, B_i\} \).

For two nonempty subsets \( X_k \) and \( X_l \) of \( V \), we define
\( \min(X_k, X_l) \) to be the shortest distance between the two cir-
cles around the bounding boxes of \( X_k \) and \( X_l \) and \( \max(X_k, X_l) \)
We require that all pairs in a WSPD are \( s \)-well-separated, where \( s \) is the separation constant of the WSPD, defined as \( \min(A_i, B_i) \geq s \cdot \max(diam(A_i), diam(B_i)) \) for all \( i \) with \( 1 \leq i \leq m \).

As shown by Callahan and Kosaraju [3, 4], for any \( V \) and any \( s > 0 \), there exists a WSPD (called the canonical WSPD) of size \( m = O(ns^d) \) [10, Lemma 9.4.5] that can be computed in \( O(n \log n + ns^d) \) time and can be represented in \( O(s^d) \) space.

We now state three properties of the WSPD. The references provided contain statements that are similar but slightly different from the properties stated here – see [1] for proofs how the following statements follow from the references.

**Observation 2.2** (Bose et al. [2, Observation 1]). For \( s = \frac{2t}{d+1} \) and for each \( i \) with \( 1 \leq i \leq m \), the greedy spanner includes at most one edge \((u,v)\) with \( (u,v) \in \{A_i, B_i\} \).

**Fact 2.3** (Narasimhan and Smid [10, Lemma 11.3.4]). Let \( \gamma \) and \( t \) be positive real numbers, and let \( \{A_i, B_i\} \) be a well-separated pair in the WSPD with length \( t(A_i, B_i) = t \).

The number of well-separated pairs \( \{A_i, B_i\} \) such that the length of the pair is in the interval \([\ell t / 2, 2t \ell]\) and at least one of \( R(A_i) \) and \( R(B_i) \) is within distance \( \gamma \ell \) of either \( R(A_i) \) or \( R(B_i) \) is less than or equal to \( 12 \cdot \Phi \left( \frac{d}{2}, \gamma \right) \) and

\[
\Phi \left( \frac{d}{2}, \gamma \right) = O \left( s^d \log n \right) \]

**Fact 2.4** (Callahan [3, Chapter 4.5]).

\[
\sum_{i=1}^{m} \min(|A_i|, |B_i|) = O \left( s^d n \log n \right)
\]

### 3. A SIMPLE GREEDY SPANNER ALGORITHM

We observe that algorithm GreedySpannerOriginal repeatedly adds the edge \((u,v)\) with minimal \(|uv|\) such that \((u,v)\) does not have a 1-path. This reformulation can be exploited to save on space usage [1].

We will now consider algorithm Lazy-Greedy, that follows this reformulation much more directly, resulting in a very simple linear space algorithm. Unfortunately, it becomes hard to analyze its running time. The algorithm uses a priority queue \( Q \) of pairs of points \((u,v)\) ordered by \(|uv|\).

It is clear that the space usage of this algorithm is \( O \left( \frac{n}{(t-1)^{d-1}} \right) \) (due to the size of \( E \)), and that \( O(n^3 \log n) \) is an upper bound on its time complexity, if we implement line 8 by doing a single Dijkstra computation sourced at \( a \).

By the reformulation above, it is also clear that the algorithm is correct.

However, it is unclear whether the algorithm becomes faster than \( O(n^3 \log n) \) through the use of the ‘caching’ involved by storing results in \( Q \) and marking points as clean or dirty. The experiments we have performed suggest the running time of the algorithm is near-quadratic instead of cubic. However, if for given \( n \) we take \( V = \{2^i \mid 1 \leq i \leq n\} \), it is easy to see that the running time of the algorithm is \( \Omega(n^3 \log n) \). This situation is the same as the one for the Improved Greedy algorithm introduced by Farshi and Godmundsson [6]; that algorithm is also approximately quadratic in all experiments, but, as proven by Bose et al. [2], has a cubic lower bound on certain inputs such as \( \{2^i \mid 1 \leq i \leq n\} \).

**Algorithm Lazy-Greedy**(\( V, t \))

1. \( E \) \( \leftarrow \) \( \emptyset \)
2. \( Q \) \( \leftarrow \) \( \{(a,b) \mid b \text{ is the nearest neighbor of } a\} \)
3. for \( a \in V \)
   4. do Mark \( a \) as clean
   5. while \( Q \) is not empty
      6. do Extract \((a,b) = \min(Q)\) from \( Q \)
      7. while \( a \) is marked as dirty
         8. do Compute the nearest neighbor without \( t \)-path \( k \) for \( a \)
         9. if such a \( k \) exists
            10. then Insert \((a,k)\) into \( Q \)
            11. Mark \( a \) as clean
      12. Extract \((a,b) = \min(Q)\) from \( Q \)
      13. Add \((a,b)\) to \( E \) and to \( Q \)
      14. for \((a,b) \in Q\)
         15. do Mark \( a \) as dirty
      16. return \( E \)

We will explain the discrepancy between the experimental results and the theoretical results by proving the following. For a given set of points \( V \) we define the spread of \( V \) as \( \Phi(V) = \max_{u,v \in V} [uv] \min_{u,v \in V} [uv] \).

**Fact 3.1.** Algorithm Lazy-Greedy and the Improved Greedy algorithm run in \( O(n^2 \log(n) \log(\Phi)) \) time, where \( \Phi \) is the spread of the input.

In experiments, \( \Phi \) is typically not large, while it is exponential in \( n \) for \( \{2^i \mid 1 \leq i \leq n\} \). If, for a family of inputs \( V_i, \Phi(V_i) = O(p(|V_i|)) \) where \( p \) is a polynomial, we say this family has polynomial spread. On such families, the running time of these two algorithms becomes \( O(n^2 \log^2 n) \). Polynomial spread is a very weak assumption that is satisfied by nearly all realistic data sets, explaining the speed of the above algorithm in experiments and showing that this speed is not an ‘accident’ of the tested data sets.

### 4. THE GREEDY SPANNER ALGORITHM FRAMEWORK

We first give some definitions and properties. We then define the type of algorithm that will fit our framework and give the main efficiency lemma. Finally, we give a detailed example application of the theorem.

#### 4.1 Partition Functions

Let \((a,b), (u,v) \in V^2\) be two pairs of points and let \( c \in \mathbb{R} \). We say that \((u,v)\) is \( c \)-close-by to \((a,b)\) if \(|au|, |bv| \leq c|ab|\) or if \(|au|, |bu| \leq c|ab|\). Obviously, if \((u,v) \in E\) lies on a \( t \)-path from \( a \) to \( b \), then \((u,v)\) is \( t \)-close-by to \((a,b)\). We say that a set \( X \subseteq V^2\) of pairs of points is \((c,c_1,c_2)\)-regular if we can pick \((a,b) \in X\) so that for all \((u,v) \in X, (u,v)\) is \( c \)-close-by to \((a,b)\) and \( c_1|ab| \leq |uv| \leq c_2|ab|\).

**Lemma 4.1.** Given a \((c,c_1,c_2)\)-regular set \( X \) and \( c_d \in \mathbb{R}^+ \), the number of greedy spanner edges \((u,v)\) that are \( t + c_d \)-close-by to some \((a,b) \in X\) and with \( \min_{(a,b) \in X} |ab| \leq |uv| \leq \max_{(a,b) \in X} |ab| \), is \( O \left( c_1^d \left( 1 + c + c_d \right)^d \frac{1}{(t-1)^{d-1}} \right) \).
We define a partition function \( P \) as a function taking a finite \( V \subseteq \mathbb{R}^d \) and returning some subset of \( \mathcal{P}(V^2) \) (where \( \mathcal{P} \) is the power set operator). We require \( \bigcup_{X \in \mathcal{P}(V)} X = V^2 \) and for all \( X, Y \in \mathcal{P}(V) \), \( X \cap Y = \emptyset \). We say that \( P \) is \((c_1, c_2, c_3)\)-regular if for all \( V \subseteq \mathcal{P}(V) \), \( X \in \mathcal{P}(V), \) \( c_1 \leq |X| \leq c_2 \), and \( c_3 \leq \ell \leq 2c_3 \).

We define the greedy spanner algorithm \( A \), which has a subroutine (or some other contiguously executed operation) \( O \) whose efficiency we are interested in, for example line 8 of algorithm Lazy-Greedy.

We say that \( A \) is incremental if it finds the edges of the greedy spanner in ascending order of length, and so builds up a graph \( G \) on the input \( V \). Given an operation \( O \) and a partition function \( P \), we can function \( f \) is a partition association if it maps executions of \( O \) on a point set \( V \) to an element of \( \mathcal{P}(V) \).

We now define \((c_4, c_5, c_6)\)-sporadic for an operation \( O \) of an incremental algorithm \( A \) with respect to a partition function \( P \) and a partition association \( f \) for \( O \). Let \( V \) be an input and let \( X \subseteq \mathcal{P}(V) \). Let \( EX \) be the set of executions of \( O \) that are mapped to \( X \) by \( f \). We require that there exists a \( EX' \subseteq EX \) with \(|EX'| + c_5 \geq |EX| \) for which the following holds. We order \( EX' = \{e_1, \ldots, e_{|EX'|}\} \) in ascending order according to the numbers of steps taken by \( A \) since it started on \( V \) until it started with that execution \( e_i \). Let \( 1 \leq i < |EX'| \) be an integer and let \( x_1 \) the first greedy spanner edge found after \( e_{i+1} \). Our requirement is that there exists some \( u, v \in V \) so that \((u, v)\) is \( c_6\)-close to some pair \((a, b) \in X \), \(|uw| \leq |x_1| \), and the shortest distance between \( u \) and \( v \) in the graph that \( A \) is building up at the time that \( e_i \) was executed is different from the shortest distance between \( u \) and \( v \) in the graph at the time that \( e_{i+1} \) is executed.

Main Lemma 4.2. Let \( A \) be an incremental algorithm with an operation \( O \), a \((c_1, c_2, c_3)\)-regular partition function \( P \) and a partition association \( f \). If \( O \) is \((c_4, c_5, c_6)\)-sporadic with respect to \( P \) and \( f \), then for any input \( V \) and \( X \subseteq \mathcal{P}(V) \), the number of operations that \( f \) associates with \( X \) is \( c_6 + O\left(c_4^2(1+c_5+c_6)^d \left|\frac{1}{(t-1)^{2d}}\right|\right) \). Furthermore, if \( T(X) \) is an upper bound on the time taken by any execution of \( O \) associated with \( X \) by \( f \), then the total time taken by \( O \) is \( O\left(c_4^2(1+c_5+c_6)^d \left|\frac{1}{(t-1)^{2d}}\right|\right) \sum_{X \subseteq \mathcal{P}(V)} T(X) \).
follows that \((a, k)\) was added as an edge, giving it a \(t\)-path. Line 8 will therefore never again output it, contradicting \(k = k'\).

We therefore choose \((u, v) = (a, k)\) for the definition. As \((a, k) \in X\), we can pick \(c_d = 1\) to satisfy that \((u, v)\) needs to be \(c_d\)-close to some element of \(X\). \(|w| \leq |x_i|\) is satisfied as the elements extracted from \(Q\) have monotonically increasing length. Lastly, as \(k \neq k'\), \((a, k)\) must have a \(t\)-path just after \(e_{i+1}\), while it did not have one just before \(e_i\), so its network distance has changed. This shows that \(O\) is (1, 1)-sporadic.

Setting \(T(X) = O\left(\frac{n \log n}{(t - 1)^{2d}}\right)\), Lemma 4.2 then gives us that the running time of \(O\) is

\[
O\left(\frac{1}{(t - 1)^{2d}} \sum_{X \in P_{SW}(V)} T(X)\right) = O\left(n \log n + \frac{n}{(t - 1)^{2d - 1}} \sum_{v \in V} |X_v| + |Y_v|\right) = O\left(n^2 \log n \log \Phi\left(\frac{1}{(t - 1)^{d + 1}}\right) + n^2 \log \Phi\left(\frac{1}{(t - 1)^{4d - 1}}\right)\right)
\]

**Theorem 4.4.** Algorithm Lazy-Greedy computes the greedy spanner in \(O\left(n^2 \log n \log \Phi\left(\frac{1}{(t - 1)^{d + 1}}\right) + n^2 \log \Phi\left(\frac{1}{(t - 1)^{4d - 1}}\right)\right)\) time while using \(O\left(\frac{n}{(t - 1)^{2d}}\right)\) space.

5. **MORE APPLICATIONS OF THE FRAMEWORK**

5.1 Improved-Greedy

We will now apply our framework to obtain time bounds on Algorithm Improved-Greedy and a variant of it, and on a variant of [1]. It was originally conjectured [6] that this algorithm had a running time of \(O(n^2 \log n)\). An example forcing a \(\Omega(n \log n)\) running time is \((2^i | 1 \leq i \leq n)\) [2]. Our framework gives the following:

**Theorem 5.1.** Algorithm Improved-Greedy computes the greedy spanner in \(O\left(n^2 \log n \log \Phi\left(\frac{1}{(t - 1)^{d + 1}}\right) + n^2 \log \Phi\left(\frac{1}{(t - 1)^{4d - 1}}\right)\right)\) time and \(O(n^2)\) space.

**Proof.** We use \(P_{SW}\) as our partition function. We associate executions of line 8 with the \(X \in P_{SW}(V)\) such that \((u, v) \in X\) (with \(u\) the variables of the algorithm). This operation is (1, 1)-sporadic as follows. We exclude from \(EX\) the \((u, v)\) pairs that end up being greedy spanner edges. Let \(e_i\) be an execution of line 8. \(e_i\) sets \(d[u, v]\) to the network distance of \((u, v)\). Then, when \(e_{i+1}\) is executed, \(d[u, v] > t \cdot |uv|\), but once \(d[u, v]\) has been updated to the current network distance by \(e_{i+1}\), \(d[u, v] \leq t \cdot |uv|\) (for \(e_{i+1}\) did not give an edge), and so the network distance of \((u, v)\) changed as required. The theorem then follows by applying Lemma 4.2 and using the analysis from Section 4.3. \(\square\)

**Algorithm Improved-Greedy**

1. \(E \leftarrow \emptyset\)
2. for \(u, v \in V\) do
3. \(d[u, v] = \infty\)
4. for \(u \in V\) do
5. \(d[u, u] = 0\)
6. for every pair of distinct points \((u, v)\) in ascending order of \(|uv|\) do
7. \(d[u, v] = t \cdot |uv|\)
8. then Perform a Dijkstra computation from \(u\) and update \(d[u, v]\) to the distances found
9. if \(d[u, v] > t \cdot |uv|\) then Add \((u, v)\) to \(E\)
10. return \(E\)

5.2 Doubled-Improved-Greedy

In [2], the authors noted that doing a Dijkstra computation not just sourced at \(u\) as per line 8, but also a Dijkstra computation sourced at \(v\), resulted in a \(O(n^2 \log n)\) running time on \((2^i | 1 \leq i \leq n)\). They gave an input for which this variation of the algorithm has a \(\Omega(n^2 \log n)\) running time. We now prove this lower bound is tight. We refer to the variation as Doubled-Improved-Greedy.

**Theorem 5.2.** Algorithm Doubled-Improved-Greedy computes the greedy spanner in \(O\left(n^2 \log^2 n \frac{1}{(t - 1)^{2d}} + n^2 \log n \frac{1}{(t - 1)^{4d - 1}}\right)\) time and \(O(n^2)\) space.

**Proof.** As operation \(O\) we pick a single Dijkstra computation. We use \(P_{SW}\) as our partition function. We pick the same partition association (extended to also associate the new Dijkstra computations) as Theorem 5.1. \(O\) is still (1, 1)-sporadic on the variation by the same argument as used above.

We will use the (stronger) formulation of Lemma 4.2 that says that \(|EX| = O(1)|\). Let \(1 \leq i \leq m\) and let \(EX_{u,i}\) be the set of executions associated with the \(X \in P_{SW}(V)\) such that \(X\) consists of pairs of points, either of which is \(u\), that are in the \(i\)-th well-separated pair. We note that \(\sum_{1 \leq i \leq m} |EX_{u,i}| = \sum_{1 \leq i \leq m} |EX_{v,i}|\), as every operation ‘triggers’ another operation on the ‘other side’ of the well-separated pair. Using that \(|EX_{u,i}| = O(1)|\) by Lemma 4.2 and using Fact 2.4, we can bound the number of operations executed by

\[
\sum_{i=1}^{m} O\left(\frac{1}{(t - 1)^{2d}}\right) \min(|A_i|, |B_i|) = O\left(n \log n \frac{1}{(t - 1)^{2d}}\right)
\]

The theorem follows. \(\square\)

5.3 WSPD-Greedy-Lazy

We will now give a variation on the algorithm from [1]. We will refer to the original as WSPD-Greedy. Instead of recomputing \(ClosestPair(j)\) for all nearby well-separated pairs, we use a strategy similar to Lazy-Greedy: all pairs are either dirty or clean. Whenever we add an edge, all pairs are marked as dirty. If we extract a dirty well-separated pair, we compute \(ClosestPair(j)\) for it, update its entry in the queue and mark it as clean. If we extract a clean pair, we add it to the queue as a greedy spanner edge. Instead of filling up the queue with pairs of similar lengths, we add all of them at the start of the algorithm (with keys computed by \(ClosestPair(j)\)).
We also change that if an edge is added, the pair the edge originated from is re-added to the queue. This allows us to compute a different WSPD with \( s = 4 \) instead of \( s = \frac{2t}{t-1} \) while maintaining correctness, which in turn reduces its space usage (not counting space needed to store all the edges found by the algorithm) from \( \frac{(t-1)^2}{(t-1)^3} \) to \( \frac{(t-1)^2}{(t-1)^2-1} \).

However, the running time of the algorithm is no longer easily analyzed.

We call this variation WSPD-Greedy-Lazy. We can analyze the running time of the algorithm using our framework:

**Theorem 5.3.** Algorithm WSPD-Greedy-Lazy computes the greedy spanner in
\[
O\left(n^2 \log^2 n \frac{1}{(t-1)^{2d}} + n^2 \log n \frac{1}{(t-1)^{d-1}}\right)
\]
time and
\[
O\left(\frac{n}{(t-1)^{d-1}}\right)
\]
space.

**Proof.** We need to bound the number of ClosestPair\((j)\) calls done by the algorithm. We use \( P_w \) as our partition function and associate an execution of \( O \) with the well-separated pair it is performed on. Let \((u_i, v_i)\) be the output of execution \( e_i \) and let \((u_{i+1}, v_{i+1})\) be the output of execution \( e_{i+1} \). We first show that \((u_i, v_i) \neq (u_{i+1}, v_{i+1})\): suppose they are equal, then \((u_i, v_i)\) would have been the lowest element of \( Q \) and be cleaned. It would immediately extracted again, still being the lowest element of \( Q \).

It would immediately extracted again, still being the lowest element of \( Q \), and then be added as an edge as it is clean. It gains a \( t \)-path, and therefore it cannot be the output of \( e_{i+1} \), reaching a contradiction. \((u_i, v_i)\) is therefore no longer the closest pair of points without \( t \)-path when \( e_{i+1} \) is executed and as edges only get added, this means it must have gained a \( t \)-path (as opposed to \((u_{i+1}, v_{i+1})\) losing a \( t \)-path), which means it must have changed network distance, making \( O(1,0) \)-sporadic.

Using Lemma 4.2 with
\[
T(X) = O\left(\left(n \log n + \frac{1}{(t-1)^{d-1}}\right) \min(|A_i|, |B_i|)\right)
\]
(with \( i \) the index of the well-separated pair corresponding to \( X \)) and Fact 2.4, we conclude that the running time of WSPD-Greedy-Lazy is
\[
O\left(\frac{1}{(t-1)^{2d}} \sum_{X \in P_w(V)} T(X)\right) =
\]
\[
O\left(\frac{1}{(t-1)^{2d}} \sum_{i=1}^{m} T(X_i)\right) =
\]
\[
O\left(\frac{1}{(t-1)^{3d}} \sum_{i=1}^{m} \left(n \log n + \frac{1}{(t-1)^{d-1}}\right) \min(|A_i|, |B_i|)\right) =
\]
\[
O\left(n^2 \log^2 n \frac{1}{(t-1)^{2d}} + n^2 \log n \frac{1}{(t-1)^{d-1}}\right)
\]
\[
\square
\]

### 5.4 Other algorithms

We note that WSPD-Greedy and the two algorithms from [2] also fit our framework. We will call the main algorithm from [2] BCFMS-Greedy. The algorithm from [1] readily fits using the well-separated pair partition function as per Theorem 5.3 – sporadicness is particularly easy as a recalculation can only occur if a greedy spanner edge was added close by, which alters the network distance of its endpoints.

We define the length-bunched partition function \( P_L \) as follows. If \( d_m = \min_{u,v \in V, u \neq v} |uv| \), then
\[
P_L(V) = \{(u, v) \mid d_m 2^i \leq |uv| \leq d_m 2^{i+1} \mid u \in V, i \in \mathbb{N}\}
\]
It is easy to see that \( P_L \) is \((2,1,2)\)-regular. We can use \( P_L \) to show that [2] fits our framework.

We associate the Dijkstra-Undo and Dijkstra-Bounded with their origin and length bucket. Again, sporadicness is easily proven as recalculations only occur if a greedy spanner edge was added nearby. One could consider a ‘lazy’ variant of this algorithm using a queue (containing the nearest neighbor without \( t \)-path as discovered by Dijkstra-Bounded)
and dirty/clean flags, in which Dijkstra-Undo and Dijkstra-Bounded are only triggered if a dirty entry is extracted from the queue. This may improve its speed in practice, but does nothing to improve space requirements.

6. EXPERIMENTAL RESULTS

We have run the algorithms discussed above on point sets whose size ranged from 500 to 128,000 points. As Improved-Greedy and Doubled-Improved-Greedy perform nearly identically, we only included Improved-Greedy in our results. We generated 2D point sets from several distributions. We have recorded space usage and running time (wall clock time). The results are averages over several runs where new point sets were generated for each run. We discuss both a time). The results are averages over several runs where new instance sizes starting at $n = 500$, doubling $n$ each time. The largest instance we could run BCFMS-Greedy on was $n = 4,000$. Improved-Greedy could reach $n = 8,000$ before running out of memory at $n = 16,000$.

On the uniform point set even the linear space WSDP-Greedy algorithm runs into trouble quite quickly for low $t$. At $n = 32,000$ it already consumes about 4GB of memory, whereas the lazy variant, WSDP-Greedy-lazy consumes only 200MB. The clear winner with respect to both memory usage and time is the Lazy-Greedy algorithm running almost 9 times as fast as WSDP-Greedy and using only 17MB.

The WSDP is usually very small on clustered point sets. Nevertheless, the Lazy-Greedy algorithm again uses a lot less memory for only roughly a factor 2 decrease in speed. For $n = 64,000$ the WSDP-Greedy algorithm uses just over 1GB, its lazy variant reduces this to 254MB and Lazy-Greedy only used 27MB.

As the dilation increases and space usage becomes less of an issue for the non-lazy linear space algorithm it becomes more competitive (but still hard to implement). Figure 2 shows that for for a high dilation of $t = 2$ the non-lazy linear space algorithm has started to become faster. The memory usage is still vastly bigger as is shown on the right. For low $t$, which is our main use case, the Lazy-Greedy algorithm again uses a lot less memory for only roughly a factor 2 decrease in speed. For $n = 64,000$ the WSDP-Greedy algorithm uses just over 1GB, its lazy variant reduces this to 254MB and Lazy-Greedy only used 27MB.

### 6.2 Dependence on instance size

We have compared running time and space usage of the algorithms discussed above for different values of $n$. We plotted the results using $t = 1.1$ on both uniform (Fig. 4) and clustered points (Fig. 5).

The Improved-Greedy and BCFMS-Greedy algorithms use quadratic memory, which makes running them on large point sets infeasible no matter which dilation is used. We used instance sizes starting at $n = 500$, doubling $n$ each time. The largest instance we could run BCFMS-Greedy on was $n = 4,000$. Improved-Greedy could reach $n = 8,000$ before running out of memory at $n = 16,000$.

On the uniform point set even the linear space WSDP-Greedy algorithm runs into trouble quite quickly for low $t$. At $n = 32,000$ it already consumes about 4GB of memory, whereas the lazy variant, WSDP-Greedy-lazy consumes only 200MB. The clear winner with respect to both memory usage and time is the Lazy-Greedy algorithm running almost 9 times as fast as WSDP-Greedy and using only 17MB.

The WSDP is usually very small on clustered point sets. Nevertheless, the Lazy-Greedy algorithm again uses a lot less memory for only roughly a factor 2 decrease in speed. For $n = 64,000$ the WSDP-Greedy algorithm uses just over 1GB, its lazy variant reduces this to 254MB and Lazy-Greedy only used 27MB.

As the dilation increases and space usage becomes less of an issue for the non-lazy linear space algorithm it becomes more competitive (but still hard to implement). Figure 2 shows that for for a high dilation of $t = 2$ the non-lazy linear space algorithm has started to become faster. The memory usage is still vastly bigger as is shown on the right. For low $t$, which is our main use case, the Lazy-Greedy algorithm again uses a lot less memory for only roughly a factor 2 decrease in speed. For $n = 64,000$ the WSDP-Greedy algorithm uses just over 1GB, its lazy variant reduces this to 254MB and Lazy-Greedy only used 27MB.

As the dilation increases and space usage becomes less of an issue for the non-lazy linear space algorithm it becomes more competitive (but still hard to implement). Figure 2 shows that for for a high dilation of $t = 2$ the non-lazy linear space algorithm has started to become faster. The memory usage is still vastly bigger as is shown on the right. For low $t$, which is our main use case, the Lazy-Greedy algorithm again uses a lot less memory for only roughly a factor 2 decrease in speed. For $n = 64,000$ the WSDP-Greedy algorithm uses just over 1GB, its lazy variant reduces this to 254MB and Lazy-Greedy only used 27MB.

As the dilation increases and space usage becomes less of an issue for the non-lazy linear space algorithm it becomes more competitive (but still hard to implement). Figure 2 shows that for for a high dilation of $t = 2$ the non-lazy linear space algorithm has started to become faster. The memory usage is still vastly bigger as is shown on the right. For low $t$, which is our main use case, the Lazy-Greedy algorithm again uses a lot less memory for only roughly a factor 2 decrease in speed. For $n = 64,000$ the WSDP-Greedy algorithm uses just over 1GB, its lazy variant reduces this to 254MB and Lazy-Greedy only used 27MB.

As the dilation increases and space usage becomes less of an issue for the non-lazy linear space algorithm it becomes more competitive (but still hard to implement). Figure 2 shows that for for a high dilation of $t = 2$ the non-lazy linear space algorithm has started to become faster. The memory usage is still vastly bigger as is shown on the right. For low $t$, which is our main use case, the Lazy-Greedy algorithm again uses a lot less memory for only roughly a factor 2 decrease in speed. For $n = 64,000$ the WSDP-Greedy algorithm uses just over 1GB, its lazy variant reduces this to 254MB and Lazy-Greedy only used 27MB.

As the dilation increases and space usage becomes less of an issue for the non-lazy linear space algorithm it becomes more competitive (but still hard to implement). Figure 2 shows that for for a high dilation of $t = 2$ the non-lazy linear space algorithm has started to become faster. The memory usage is still vastly bigger as is shown on the right. For low $t$, which is our main use case, the Lazy-Greedy algorithm again uses a lot less memory for only roughly a factor 2 decrease in speed. For $n = 64,000$ the WSDP-Greedy algorithm uses just over 1GB, its lazy variant reduces this to 254MB and Lazy-Greedy only used 27MB.

As the dilation increases and space usage becomes less of an issue for the non-lazy linear space algorithm it becomes more competitive (but still hard to implement). Figure 2 shows that for for a high dilation of $t = 2$ the non-lazy linear space algorithm has started to become faster. The memory usage is still vastly bigger as is shown on the right. For low $t$, which is our main use case, the Lazy-Greedy algorithm again uses a lot less memory for only roughly a factor 2 decrease in speed. For $n = 64,000$ the WSDP-Greedy algorithm uses just over 1GB, its lazy variant reduces this to 254MB and Lazy-Greedy only used 27MB.

As the dilation increases and space usage becomes less of an issue for the non-lazy linear space algorithm it becomes more competitive (but still hard to implement). Figure 2 shows that for for a high dilation of $t = 2$ the non-lazy linear space algorithm has started to become faster. The memory usage is still vastly bigger as is shown on the right. For low $t$, which is our main use case, the Lazy-Greedy algorithm again uses a lot less memory for only roughly a factor 2 decrease in speed. For $n = 64,000$ the WSDP-Greedy algorithm uses just over 1GB, its lazy variant reduces this to 254MB and Lazy-Greedy only used 27MB.

As the dilation increases and space usage becomes less of an issue for the non-lazy linear space algorithm it becomes more competitive (but still hard to implement). Figure 2 shows that for for a high dilation of $t = 2$ the non-lazy linear space algorithm has started to become faster. The memory usage is still vastly bigger as is shown on the right. For low $t$, which is our main use case, the Lazy-Greedy algorithm again uses a lot less memory for only roughly a factor 2 decrease in speed. For $n = 64,000$ the WSDP-Greedy algorithm uses just over 1GB, its lazy variant reduces this to 254MB and Lazy-Greedy only used 27MB.

As the dilation increases and space usage becomes less of an issue for the non-lazy linear space algorithm it becomes more competitive (but still hard to implement). Figure 2 shows that for for a high dilation of $t = 2$ the non-lazy linear space algorithm has started to become faster. The memory usage is still vastly bigger as is shown on the right. For low $t$, which is our main use case, the Lazy-Greedy algorithm again uses a lot less memory for only roughly a factor 2 decrease in speed. For $n = 64,000$ the WSDP-Greedy algorithm uses just over 1GB, its lazy variant reduces this to 254MB and Lazy-Greedy only used 27MB.
Figure 4: The left plot shows the running time for $t = 1.1$ on variously sized uniformly distributed instances. The right plot shows the memory usage on the same data.

Figure 5: The left plot shows the running time for $t = 1.1$ on variously sized clustered instances. The right plot shows the memory usage on the same data.

7. CONCLUSION

We have presented our framework of an abstract greedy spanner algorithm satisfying certain properties and have shown that this implies that a part of this algorithm will then be executed only few times. We have shown that many algorithms fit this framework, giving many new subcubic bounds, both on old and on new algorithms introduced in this paper.

In practice, the novel algorithm we present enormously decreases memory usage over the previous state-of-the-art algorithm. For low values of $t$, it is also significantly faster, while never being much slower even for high $t$. This both enables larger point sets to have their greedy spanner calculated, and enables lower values of $t$ to be used. The new algorithm is also very simple to implement.

Our results could be improved in several ways. One could try to remove some log $n$ factors from our algorithms or from our analysis, one could try to reduce the dependency on $t$ or change a dependency on log $\Phi$ to log $n$. A general sub-quadratic time algorithm remains elusive. It might also be interesting to generalize our results to more general metric spaces.

8. REFERENCES


