Constructing Minimum-Interference Networks*

Marc Benkert† Joachim Gudmundsson‡ Herman Haverkort§
Alexander Wolff¶

Submitted to CGTA: January 3, 2007. Revised: May 16, 2007.

Abstract

A wireless ad-hoc network can be represented as a graph in which the nodes represent wireless devices, and the links represent pairs of nodes that communicate directly by means of radio signals. The *interference* caused by a link between two nodes u and v can be defined as the number of other nodes that may be disturbed by the signals exchanged by u and v. Given the position of the nodes in the plane, links are to be chosen such that the maximum interference caused by any link is limited and the network fulfills desirable properties such as connectivity, bounded dilation or bounded link diameter. We give efficient algorithms to find the links in two models. In the first model, the signal sent by u to v reaches exactly the nodes that are not farther from u than v is. In the second model, we assume that the boundary of a signal's reach is not known precisely and that our algorithms should therefore be based on acceptable estimations. The latter model yields faster algorithms.

1 Introduction

Wireless ad-hoc networks consist of a number of wireless devices spread across a geographical area. Each device has wireless communication capability, some level of intelligence for signal processing and networking of the data, and a typically limited power source such as a small battery.

*This article is based on the preliminary version [M. Benkert, J. Gudmundsson, H. Haverkort, A. Wolff: Constructing Interference-Minimal Networks, in: *Proc. 32nd Intern. Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM'06)*, pages 166–176, in: Lecture Notes in Computer Science 3831, Springer-Verlag, Berlin, 2006]. Our work has been supported by grant WO 758/4-2 of the German Research Foundation (DFG) and by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council. Part of this research was done while H. Haverkort was working at Karlsruhe University, supported by the European Commission, FET open project DELIS (IST-001907).

†Department of Computer Science, Karlsruhe University, P.O. Box 6980, D-76128 Karlsruhe, Germany. WWW: `http://i11www.iti.uni-karlsruhe.de/members/mbenkert`

‡National ICT Australia Ltd, Sydney, Australia. Email: `joachim.gudmundsson@nicta.com.au`

§Department of Computer Science, TU Eindhoven, The Netherlands. Email: `cs.herman@haverkort.net`

¶Department of Computer Science, TU Eindhoven, The Netherlands. WWW: `http://www.win.tue.nl/~awolff`

This paper studies networks that do not depend on dedicated base stations: in theory, all nodes may communicate directly with each other. In practice however, this is often a bad idea: if nodes that are far from each other would exchange signals directly, their signals may interfere with the communication between other nodes within reach. This may cause errors, so that messages have to be sent again. Communicating directly over large distances would also require sending very strong signals, since the necessary strength depends at least quadratically on the distance (in practice the dependency tends to be cubic or worse). Both issues could lead to rapid depletion of the devices' limited power sources. Therefore it is advisable to organize communication between nodes such that direct communication is restricted to pairs of nodes that can reach each other with relatively weak signals that will only disturb a small number of other nodes. We model such a network as a graph $G = (V, E)$, in which the vertices V represent the positions of the mobile devices in the plane, and the links (or edges) E represent the pairs of nodes that exchange signals directly. Communication between nodes that do not exchange signals directly should be routed over other nodes on a path through that network. According to Prakash [Pra99], the basic communication between direct neighbors becomes unacceptably problematic if the acknowledgement of a message is not sent on the same link in opposite direction. Therefore we will assume that the links are undirected.

Our problem is therefore to find an undirected graph on a given set of nodes in the plane, such that all nodes are connected with each other through the network (preferably over a short path), interference problems are minimized, and direct neighbors in the network can reach each other with signals of bounded transmission radius. In this paper we focus on guaranteeing connectivity and minimizing interference; bounding the transmission radius is an easy extension, which we discuss in Section 4. Since wireless devices tend to move frequently, we need to be able to construct networks with the desired properties fast.

The optimal network structure depends ultimately on the actual communication that takes place. This is generally not known a priori. Therefore we aim to optimize a network property that we believe to be a good indicator of the probability that interference problems will occur. Assuming that each node can adjust the strength of each signal so that it can just reach the intended receiver, such indicators may be:

sending-link-based interference of a link $\{u, v\}$: the number of nodes that are within reach of the signals from the communication over a particular link $\{u, v\}$ in the network (proposed by Burkhart et al. [BvRWZ04], also studied by Moaveni-Nejad and Li [ML05]) – in other words, the number of nodes that are hindered when the link $\{u, v\}$ is active. This is the definition of interference we focus on in this paper.

sending-node-based interference of a node u: the number of nodes that receive signals transmitted by u (proposed by Moaveni-Nejad and Li [ML05]) – in other words, the number of nodes that are hindered when u is active.

receiving-node-based interference of a node u: the number of nodes transmitting signals that reach u (proposed by Rickenbach et al. [vRSWZ05]) – that is, the number of nodes that may prevent u from communicating effectively.

Previous results. To construct a network that connects all nodes and minimizes the maximum and total sending-link-based interference, we could run Prim's minimum-spanning-tree algorithm [Pri57] with a Fibonacci heap. Assuming that the interference for each feasible link is given in advance, this takes $O(m + n \log n)$ time, where n is the number of nodes and m is the number of eligible links. If all possible links are considered, we can compute their interference values in $O(n^{9/4} \operatorname{polylog} n)$ time (see the proof of Lemma 3). This will then dominate the total running time.

To make sure that nodes are connected by a relatively short path in the network, one could construct a t-spanner on the given set of nodes. A network G is said to be a t-spanner if, for every pair of vertices u and v, the length of the shortest path in the network is at most a chosen constant t times the Euclidean distance between u and v. The *dilation* of a graph G is the smallest t such that G is a t-spanner. Burkhart et al. [BvRWZ04] presented a first algorithm to construct a t-spanner for given $t > 1$ such that the maximum interference of the edges in the spanner is minimized. It was later improved by Moaveni-Nejad and Li in [ML05]. Assuming that the interference for each possible link is again given in advance, the running time of their algorithm is $O(n \log n(m + n \log n))$. If all possible links are considered, the running time is $O(n^3 \log n)$.

The approach for sending-linked-based interference can be modified to optimize sending-node-based interference by defining the interference of a link $\{u, v\}$ to be the maximum of the sending-node-based interferences of u and v. The maximum occurring interference of a link will then be the maximum occurring node interference in the original sending-node-based setting. Unfortunately, we cannot apply this modification to fit the receiving-node-based interference. With sending-link-based interference we can decide whether a link causes too much interference independently of the other links that may be active. With receiving-node-based interference this is not possible, so that completely different algorithms would be needed. Rickenbach et al. [vRSWZ05] only give an approximation algorithm for the case where all nodes are on a single line (the *highway model*).

Our results. We improve and extend the results of Burkhart et al. and Moaveni-Nejad and Li in two ways.

First, apart from considering networks that are simply connected (spanning trees) and networks with bounded dilation (t-spanners), we also consider networks with bounded link diameter, that is, networks such that for every pair of nodes $\{u, v\}$ there is a path from u to v that consists of at most d links (or 'hops'), where d is a parameter given as input to the algorithm. Such d-hop networks are useful since much of the delay while sending signals through a network is typically time spent by signals being processed in the nodes rather than time spent by signals actually traveling.

Second, we remove the assumption that the interference of each possible link is given in advance. For each of the three properties (connectivity, bounded dilation or bounded link diameter), we present algorithms that decide whether the graph G_k with all links of interference at most k has the desired property. The main idea is that we significantly restrict the set of possible links for which we have to determine the interference, in such a way that we can still decide correctly whether G_k has the desired property. To find the smallest k such that

	exact model (expected running times)	estimation model (deterministic)
spanning tree	$\min\{n^{9/4}\operatorname{polylog} n, nk^2 + nk\log n\}$	$\frac{n}{\varepsilon^2}\left(\frac{1}{\varepsilon} + \log n\right)$
t-spanner	$n^2 \log k(k + \log n)$	$n^2 \log k\left(\frac{1}{\varepsilon^2} + \log n\right)$
d-hop network	$\min\{n^{9/4}\operatorname{polylog} n, nk^2\} + n^2 \log n \log k$	$\frac{n}{\varepsilon^2}\left(\frac{1}{\varepsilon} + n \log k\right)$

Table 1: Running times of our algorithms to find a minimum-interference network with the required property. The running times are given in big-Oh notation, n is the number of nodes, k is the maximum interference of any link in the resulting network, and ε specifies the relative inaccuracy with which a signal's reach and the dilation of a spanner are known. Worst-case running times for deterministic algorithms for the exact model are slightly worse than the expected running times of the randomized algorithms listed in the table. The listed running times for the estimation model are worst-case.

there is a network with interference k and the desired property, we do a combined exponential and binary search that calls the decision algorithm $O(\log k)$ times. The resulting algorithms are output-sensitive: their running times depend on the interference of the resulting network.

Our algorithms work for sending-link-based and sending-node-based interference. We present two models: the *exact* model and the *estimation* model. In the exact model, we assume that the signal sent by a node u to a node v reaches exactly the nodes that are not farther from u than v is. Our algorithms for this model are faster than the algorithm by Moaveni-Nejad and Li [ML05] for $k \in o(n)$. In the estimation model, we assume that it is not realistic that the boundary of a signal's reach is known precisely: for points w that are slightly farther from u than v is, counting w as being disturbed by the signal sent from u to v is as good a guess as not counting w as disturbed. It turns out that with this model, the number of links for which we actually have to compute the interference can be reduced much further, so that we get faster algorithms, especially for spanning trees with larger values of k. Our results are listed in Table 1.

This paper is organized as follows. In Section 2 we propose our output-sensitive algorithms for the case of exact interference values, and examine their running times. In Section 3 we introduce our model for estimations of link interference and describe one algorithm for each of the network properties (connectivity, bounded dilation, and bounded link diameter). In Section 4 we briefly discuss some generalizations of our algorithms: nodes with bounded transmission radius and weighted nodes.

2 Computing exact-interference graphs

We are given a set V of n points in the plane in general position, that is, we assume that no three points lie on a straight line or a circle[1]. Our aim is to establish a wireless network that minimizes interference. First, we define interference. Let u, v be any two points in V. If the edge (link) $\{u, v\}$ is contained in a communication network, the range of u must be at least $|uv|$. Hence, if u sends a signal to v this causes interferences within the closed disk $D(u, |uv|)$ that has center u and radius $|uv|$. The same holds for v. This leads to the following definition that was first given by Burkhart et al. [BvRWZ04]. See also Figure 1.

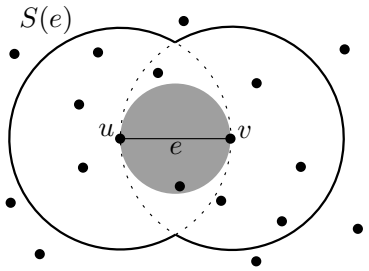

Fig. 1: Illustrating the sphere $S(e)$ of e. Here $\mathrm{Int}(e) = 9$.

Definition 1 ([BvRWZ04]) *Let* $\binom{V}{2}$ *denote the set of all pairs* $\{u, v\} \in V$ *with* $u \neq v$. *The* sphere *of an edge* $e = \{u, v\}$ *is defined as* $S(e) := D(u, |uv|) \cup D(v, |uv|)$. *For any edge* $e = \{u, v\} \in \binom{V}{2}$ *we define the* interference *of* e *by* $\mathrm{Int}(e) := \left|V \cap S(e) \setminus \{u, v\}\right|$. *The* interference $\mathrm{Int}(G)$ *of a graph* $G = (V, E)$ *is defined by* $\mathrm{Int}(G) := \max_{e \in E} \mathrm{Int}(e)$.

In this section, we will give algorithms to compute minimum-interference networks of three types. The first type is a spanning tree $\mathcal{T}$, the second type of network is, for an additionally given $t \geq 1$, a t-spanner, and the third type is a d-hop network, for a given integer $d > 1$.

The main idea of the algorithms is the same. For given $j \geq 0$ let $G_j = (V, E_j)$ denote the graph[2] where E_j includes all edges e with $\mathrm{Int}(e) \leq j$. Assume that E_j can be computed by calling a subroutine *ComputeEdgeSet* with arguments V and j. Exponential and binary search are used to determine the minimum value of k for which G_k has the desired property $\mathcal{P}$, see Algorithm 1. We first try $upper = 0$, and compute all edges of G_0. If G_0 does not have the desired property, we continue with $upper = 1$ and then keep doubling $upper$ until G_{upper} has the desired property. We compute the interference values for each of its edges, and continue with a binary search between $lower = upper/2$ and $upper$. In each step we construct G_{middle}, the graph to be tested, by selecting the

[1]None of the algorithms presented in this paper explicitly requires the point set to be in general position, however, some of the tools used assume general position, for example the concept of higher-order Delaunay edges and the range searching data structures used in Section 2.1.

[2]If the sphere of an edge is defined as $D(u, |uv|) \cap D(v, |uv|)$ then G_j is the j-relative neighbourhood graph, and if the sphere is the disk $D((u + v)/2, |uv|/2)$ (shaded in Figure 1) then G_j is the j-Gabriel graph [JT92].

Algorithm 1 MinInterferenceNetwork$(V, \mathcal{P})$

```
// exponential search
upper ← 1/4
repeat
  upper ← 2 · upper
  E_upper ← ComputeEdgeSet(V, ⌊upper⌋)
  G_upper ← (V, E_upper)
until FulfillsProperty(G_upper, P)
lower ← ⌊upper/2⌋
// binary search
while upper > lower + 1 do
  middle ← (1/2)(lower + upper)
  G_middle ← (V, all edges in E_upper with interference at most middle)
  if FulfillsProperty(G_middle, P) then upper ← middle
  else lower ← middle
return G_upper
```

edges with interference at most $middle = \frac{1}{2}(lower + upper)$ from G_{upper}, which has already been computed. Note that in the binary search phase, the search interval boundaries *lower* and *upper* are always even.

To get a spanning tree, we test with the property $\mathcal{P}$ = connectivity. After running Algorithm MinInterferenceNetwork to find the minimum k for which G_k is connected, we run a standard minimum spanning tree algorithm on G_k. The result is a tree $\mathcal{T}$ that minimizes both $\max_{e\in\mathcal{T}} \mathrm{Int}(e)$ and $\sum_{e\in\mathcal{T}} \mathrm{Int}(e)$ among all spanning trees. For the t-spanner and the d-hop network, the test consists of determining the dilation or the link diameter of the network. We do this with an all-pairs-shortest-paths computation.

Note that the only non-trivial steps in algorithm MinInterferenceNetwork are the subroutines *FulfillsProperty* and *ComputeEdgeSet*. We first give details on how to implement *ComputeEdgeSet*, that is, how to compute E_j and the interference values of the edges in E_j efficiently for any j.

2.1 Computing edge interferences

An edge $\{u, v\}$ is an *order-j Delaunay edge* if there exists a circle through u and v that has at most j points of V inside [GHvK02].

Lemma 1 *All edges in E_j are order-j Delaunay edges.*

Proof. Let $e = \{u, v\}$ be an edge with $\mathrm{Int}(e) \le j$. Then by Definition 1, the sphere $S(e)$ contains at most j points other than u and v. The disk that has e as diameter and u and v on its boundary, see Figure 1, contains at most j points in its interior as it is contained in $S(e)$ and does not contain u and v. Thus, e is an order-j Delaunay edge. □

There is a close connection between order-j Delaunay edges and higher-order Voronoi diagrams that we will use.

Lemma 2 *(Lemma 2 in [GHvK02])*
Let V be a set of n points in the plane, let $j \le n/2 - 2$, and let $u, v \in V$. The

edge $\{u, v\}$ is an order-j Delaunay edge if and only if there are two incident faces, F_1 and F_2, in the order-$(j+1)$ Voronoi diagram such that u is among the $j+1$ points closest to F_1 but not among the $j+1$ points closest to F_2, while v is among the $j+1$ points closest to F_2 but not among those closest to F_1.

Since the worst-case complexity of the order-$(j+1)$ Voronoi diagram is $O((j+1)(n-j-1))$ [Lee82], it follows that $O(nj)$ pairs of points give rise to all order-j Delaunay edges. This is because any two incident faces induce exactly one point pair that corresponds to a Delaunay edge. These pairs can be computed in $O(nj2^{c\log^* j} + n\log n)$ expected time [Ram99] (the best-known deterministic algorithm has a worst-case running time that is only worse by a polylogarithmic factor [Cha00]). Note that this also implies that the number of edges in E_j is bounded by $O(nj)$.

Lemma 3 *Given n points in the plane, (i) any edge set E_j with $j \leq n/2-2$ can be computed in $O(nj^2 + nj\log n)$ expected time; (ii) after $O(n^{9/4}\,\mathrm{polylog}\,n)$ preprocessing time, any edge set E_j can be computed in $O(nj)$ worst-case time.*

Proof. (i) Computing the order-$(j+1)$ Voronoi diagram and thus all $O(nj)$ order-j Delaunay edges takes $O(nj\log j + n\log n)$ expected time. A data structure of the order-$(j+1)$ Voronoi diagram can be built in $O(nj\log n)$ time that supports point location queries in $O(\log n)$ time. Thus, the time for the construction of the diagram (and the point-location data structure) dominates the time we need to determine for each node $v \in V$ the cell in the order-$(j+1)$ Voronoi diagram in which it lies. This cell corresponds to a list L_v of the $j+1$ nearest nodes to v in V.

According to Lemma 1, E_j is contained in the set of all order-j Delaunay edges. We simply test for each of the $O(nj)$ candidate edges in E_j whether its interference is at most j. For a candidate edge $e = \{u, v\}$ we look at the lists L_u and L_v that we have precomputed. If v is not included in L_u we conclude $\mathrm{Int}(e) \geq j+1$ and reject e. Analogously, we reject e if u is not included in L_v. Otherwise we traverse L_u until we find v and count the number of points found. After that we traverse L_v until we find u and count the number of points that are not in $D(u, |uv|) \cap D(v, |uv|)$ as these points have already been counted. We accept e as an edge of E_j if and only if we count at most j points.[3] Since the lists L_u and L_v are given in advance this test can be done in $O(j)$ time per edge. Thus, testing all $O(nj)$ candidate edges requires $O(nj^2)$ time and hence computing one edge set E_j takes $O(nj^2 + nj\log n)$ expected time in total.

(ii) Project all input points (x, y) parallel to the z-axis on a three-dimensional parabola $z = x^2 + y^2$, and build a data structure on them so that we can report the number of points lying inside the intersection of two halfspaces efficiently. We use the structure by Matoušek [Mat93], who showed that one can build such a data structure of size $O(M)$ in time $O(n^{1+\delta} + M\,\mathrm{polylog}\,n)$, such that queries can be answered in $O((n/M^{1/3})\,\mathrm{polylog}\,n)$ time, where δ is an arbitrarily small positive constant. We choose $M = n^{9/4}$, and get preprocessing time $O(n^{9/4}\,\mathrm{polylog}\,n)$ and query time $O(n^{1/4}\,\mathrm{polylog}\,n)$. Observe that in the plane, an input point lies inside a circle if and only if in three dimensions, its projection

[3] Note that the computation of the order-j Voronoi diagram instead of the order-$(j+1)$ Voronoi diagram would not be sufficient to decide whether $\mathrm{Int}(e) \leq j$. For example, the nearest neighbor of u could be v while u is not one of the j nearest neighbors of v. Then we would have $\mathrm{Int}(e) \geq j$, but we cannot decide whether $\mathrm{Int}(e) = j$.

on the parabola lies below the plane that contains the projection of the circle on the parabola. The number of points in $D(u, |uv|)$ and $D(v, |uv|)$ can thus be determined by halfspace queries in the data structure mentioned above, and the number of points lying in $D(u, |uv|) \cap D(v, |uv|)$ is determined by querying the data structure with the intersection of two halfspaces.

We use this to compute $\mathrm{Int}(e)$ for all $O(n^2)$ possible edges e in $O(n^{9/4} \operatorname{polylog} n)$ total query time, sort the results for all edges by increasing interference value in $O(n^2 \log n)$ time, and store the results. After that, any edge set E_j can be found in $O(nj)$ time by simply selecting the edges e with $\mathrm{Int}(e) \leq j$ from the head of the sorted list. □

2.2 The total running time

Theorem 1 *Algorithm* MinInterferenceNetwork *runs in* $O(\min\{nk^2 + nk \log n, n^{9/4} \operatorname{polylog} n\} + P(n, nk) \log k)$ *expected time, where* n *is the number of nodes,* k *is the interference of the network output, and* $P(n, m)$ *is the running time of FulfillsProperty on a graph with* n *nodes and* m *edges.*

Proof. During the exponential-search phase, *ComputeEdgeSet* is called $\lceil \log k \rceil$ times to compute an edge set E_j in time $O(nj^2 + nj \log n)$. As the $j's$ in the running time are geometrically increasing values, the terms in the running time that depend on j are dominated by the computation of E_k. The total expected time spent by *ComputeEdgeSet* is thus $O(nk^2 + nk \log n)$ (by Lemma 3 (i)). Once the total time spent by *ComputeEdgeSet* has accumulated to $\Omega(n^{9/4} \operatorname{polylog} n)$, we compute the interference values for all possible edges at once and sort them in $O(n^{9/4} \operatorname{polylog} n)$ time. After this, we can identify all sets $E_{\lfloor upper \rfloor}$ for geometrically increasing values of *upper* up to at most $2k$ easily in $O(nk)$ time (by Lemma 3 (ii)). In the binary-search phase, selecting edges of E_{middle} from E_{upper} takes $O(nk)$ time, which is done $O(\log k)$ times for a total of $O(nk \log k)$. A total of $O(\log k)$ tests for the property $\mathcal{P}$ on graphs with $O(nk)$ edges takes $O(P(n, nk) \cdot \log k)$ time. □

2.3 Minimum-interference spanning trees

In this section we consider the basic problem of computing a connected graph with minimum interference. For a graph with n nodes and m edges, we can test by breadth-first search in $O(n + m)$ worst-case time whether it is connected. We have $P(n, nk) = O(nk)$ and by applying Theorem 1 we get:

Corollary 1 *We can compute a connected graph with minimum-interference in expected time* $O(\min\{nk^2 + nk \log n, n^{9/4} \operatorname{polylog} n\})$*, where* k *is the interference of the network output.*

We can compute the minimum spanning tree of the resulting graph in $O(nk + n \log n)$ time (for example with Prim's algorithm), with the weights of the edges set to their interference values. Note that any minimum spanning tree $\mathcal{T}$ on the given set of vertices not only minimizes $\sum_{e \in \mathcal{T}} \mathrm{Int}(e)$ but also $\mathrm{Int}(\mathcal{T}) = \max_{e \in \mathcal{T}} \mathrm{Int}(e)$ for the set of all connected trees. Therefore such a tree can be found in the minimum-interference graph G_k obtained by Corollary 1. By running Prim's algorithm on G_k, we can compute $\mathcal{T}$ from G_k in $O(nk + n \log n)$ time, i.e., $P(n, nk) = O(nk + n \log n)$. We get the following corollary:

Corollary 2 *We can compute a spanning tree $\mathcal{T}$ of V with minimum-interference that minimizes $\sum_{e\in\mathcal{T}} \mathrm{Int}(e)$ and $\max_{e\in T} \mathrm{Int}(e)$ in expected time $O(\min\{nk^2 + nk\log n, n^{9/4}\,\mathrm{polylog}\, n\})$.*

2.4 Minimum-interference t-spanners

It is often desired that a network is not only a connected network, but also a t-spanner for some given constant t [BvRWZ04, LW04]. Given two vertices u and v of a graph G, we denote by $|uv|$ the Euclidean distance between u and v and by $d_G(u,v)$ the length of a shortest path between u and v in G (with respect to the Euclidean metric). A graph G is a t-spanner, if for any $(u,v) \in \binom{V}{2}$ it holds that $d_G(u,v) \leq t \cdot |uv|$. For a graph with n nodes and m edges, the dilation can be computed in $O(nm + n^2 \log n)$ time by computing all pairs' shortest paths with Dijkstra's algorithm [Dij59]. We have $P(n, nk) = O(n^2 k + n^2 \log n)$ and by applying Theorem 1 we get:

Corollary 3 *For any $t > 1$ we can compute a t-spanner of V with minimum-interference in $O(n^2 \log k(\log n + k))$ expected time.*

The weight of a graph G, denoted $wt(G)$, is defined as the sum of its edge lengths. By adding a postprocessing step to the computation of a minimum-interference t-spanner, we obtain the following:

Lemma 4 *For any constants $t > 1$ and $\varepsilon > 0$ we can compute a $(1+\varepsilon)t$-spanner of V with weight $O(wt(MST(V))$ and interference k in $O(n^2 \log k(\log n + k) + \frac{nk}{\varepsilon^7}\log n)$ expected time, where $MST(V)$ is a minimum weight spanning tree of V and k is the interference of the minimum-estimated interference of any t-spanner of V.*

Proof. Compute a t-spanner G of V with minimum-interference using Corollary 3, and let k be the interference of G. Note that G has $O(nk)$ edges. Now we argue that we can compute a $(1+\varepsilon)$-spanner G' of G in $O(\frac{nk}{\varepsilon^7}\log n)$ time such that G' has interference k and weight $O(wt(MST(V)))$.

Let G be an arbitrary t-spanner for a point set V having m edges, where $t > 1$ is a constant. Let ε be an arbitrary positive real constant. The greedy algorithm in [GLN02] computes a subgraph G' of G that is a $(1+\varepsilon)$-spanner of G and that satisfies the so-called leapfrog property. The graph G' can be computed in $O(\frac{m}{\varepsilon^7}\log n)$ time, for details see the book by Narasimhan and Smid [NS07]. Das and Narasimhan [DN97] have shown that any set of edges that satisfies the leapfrog property has weight $O(wt(MST(V)))$. This completes the proof of the lemma. □

2.5 Minimum-interference d-hop networks

We can test whether a given graph is a d-hop network by computing its link diameter and checking if it is at most d. For a graph with n nodes and m edges, the link diameter can be computed in $O(nm)$ worst-case time by doing a breadth-first search from every vertex. Alternatively, since all edge weights for the distance computation are '1' we can compute the link diameter in $O(n^2 \log n)$ expected time with the all-pairs-shortest-paths algorithm by Moffat

and Takaoka [MT87]. Thus, $P(n, nk) = O(n^2 \log n)$ and by applying Theorem 1 we get:

Corollary 4 *For any integer $d > 1$ we can compute a d-hop network with minimum-interference in $O(\min\{nk^2, n^{9/4} \operatorname{polylog} n\} + n^2 \log n \log k)$ expected time.*

3 Estimated interference

In this section we show how to compute minimum-interference networks in the estimated model. We define estimated interference as follows (see Fig. 2a):

Definition 2 *Let $D(u, r)$ be the closed disk centered at u with radius r. The $(1+\varepsilon)$-sphere $S_{1+\varepsilon}(e)$ of an edge $e = \{u, v\}$ is defined as $S_{1+\varepsilon}(e) := D(u, (1+\varepsilon) \cdot |uv|) \cup D(v, (1+\varepsilon) \cdot |uv|)$. For $0 \leq \varepsilon_1 \leq \varepsilon_2$ we say that an integer I is an $(\varepsilon_1, \varepsilon_2)$-valid estimation of the interference of e if and only if $\left|V \cap S_{1+\varepsilon_1}(e) \setminus \{u, v\}\right| \leq I \leq \left|V \cap S_{1+\varepsilon_2}(e) \setminus \{u, v\}\right|$.*

We will use *ε-valid estimation* as a shorthand for *$(0, \varepsilon)$-valid estimation*. Our aim is to compute minimum-interference networks based on ε-valid estimations of interference. To do so we will need to fix a particular assignment $\mathrm{Int}_\varepsilon : \binom{V}{2} \to \mathbb{N}$ of ε-valid estimations for all edges, which will be explained in more detail below.

3.1 The well-separated pair decomposition

Our construction uses the well-separated pair decomposition by Callahan and Kosaraju [CK95], which we briefly present here.

Definition 3 ([CK95]) *Let $s > 0$ be a real number, and let A and B be two finite sets of points in $\mathbb{R}^2$. We say that A and B are* well-separated *with respect to s, if there are two disjoint disks D_A and D_B of same radius r, such that*
(i) D_A contains A,
(ii) D_B contains B, and
(iii) the minimum distance between D_A and D_B is at least $s \cdot r$.

The parameter s will be referred to as the *separation constant*. The next lemma follows easily from Definition 3.

Lemma 5 ([CK95]) *Let A and B be two finite sets of points that are well-separated with respect to s, let u and x be points of A, and let v and y be points of B. Then*
(i) $|uv| \leq (1 + 2/s) \cdot |uy|$,
(ii) $(1 - 4/s) \cdot |xy| \leq |uv| \leq (1 + 4/s) \cdot |xy|$, and
(iii) $|ux| \leq (2/s) \cdot |xy|$.

Definition 4 ([CK95]) *Let V be a set of n points in $\mathbb{R}^2$, and let $s > 0$ be a real number. A* well-separated pair decomposition *(WSPD) for V with respect to s is a sequence of pairs of non-empty subsets of V, $\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_m, B_m\}$, such that (i) A_i and B_i are well-separated with respect to s, for $i = 1, \ldots, m$, and*

(ii) for any two distinct points u and v of V, there is exactly one pair $\{A_i, B_i\}$ in the sequence, such that (a) $u \in A_i$ and $v \in B_i$, or (b) $v \in A_i$ and $u \in B_i$. The integer m is called the size *of the WSPD.*

Callahan and Kosaraju [CK95] showed that a WSPD of size $O(s^2 n)$ can be computed in $O(s^2 n + n \log n)$ time.

3.2 A sparse interference-estimation graph

Consider the complete graph $G^\varepsilon = (V, E^\varepsilon)$, where the weights of the edges are computed as follows.

Let $\{A_i, B_i\}_{1 \le i \le m}$ be a well-separated pair decomposition of V with separation constant $s = 8 + 18/\varepsilon$. For each well-separated pair $\{A_i, B_i\}$ select two arbitrary points $u \in A_i$ and $v \in B_i$, and compute an $(\frac{1}{3}\varepsilon, \frac{2}{3}\varepsilon)$-valid interference estimation of $\{u, v\}$. This value is assigned to all edges $\{x, y\}$ in E^ε with $x \in A_i$ and $y \in B_i$. This assignment with interference estimations for all edges $e \in E^\varepsilon$ is denoted by $\mathrm{Int}_\varepsilon(e)$.

We will denote by $G^\varepsilon_j = (V, E^\varepsilon_j)$ the subgraph of G^ε containing all the edges of weight at most j. The following lemma shows that Int_ε correctly represents ε-valid estimations of interference.

Lemma 6 *Let V be a set of points and let $\{A_i, B_i\}_{1 \le i \le m}$ be a WSPD of V with separation constant $s = 8 + 18/\varepsilon$. Let $\widetilde{e} = \{u, v\}$ and $e = \{x, y\}$ be two edges such that $u, x \in A_i$ and $v, y \in B_i$. It holds that every $(\frac{1}{3}\varepsilon, \frac{2}{3}\varepsilon)$-valid interference estimation for $\widetilde{e}$, denoted I, is an ε-valid interference estimation for e.*

Proof. We will prove the lemma in two steps. First we show (i) that $D(x, |xy|) \subseteq D(u, (1 + \frac{1}{3}\varepsilon)|uv|)$ (and analogously $D(y, |xy|) \subseteq D(v, (1 + \frac{1}{3}\varepsilon)|uv|)$) which implies that $S_1(e) \subseteq S_{1+\varepsilon/3}(\widetilde{e})$, and thus $|V \cap S_1(e)| \le |V \cap S_{1+\varepsilon/3}(\widetilde{e})| \le I$. Then, in the second step, we show (ii) $D(u, (1 + \frac{2}{3}\varepsilon)|uv|) \subseteq D(x, (1 + \varepsilon)|xy|)$ (and analogously $D(v, (1 + \frac{2}{3}\varepsilon)|uv|) \subseteq D(y, (1 + \varepsilon)|xy|)$) which implies that $S_{1+2\varepsilon/3}(\widetilde{e}) \subseteq S_{1+\varepsilon}(e)$, and thus $I \le |V \cap S_{1+2\varepsilon/3}(\widetilde{e})| \le |V \cap S_{1+\varepsilon}(e)|$. As a consequence of the two bounds we have $|V \cap S_1(e)| \le I \le |V \cap S_{1+\varepsilon}(e)|$ and thus I is an ε-valid interference estimation for e.

(i): Let p_x be the point on the perimeter of $D(u, (1 + \frac{1}{3}\varepsilon)|uv|)$ closest to x. It suffices to show that $|xp_x| \ge |xy|$. We use the triangle inequality and Lemma 5(ii) and (iii).

$$\begin{aligned}
|xp_x| &\ge |p_x u| - |xu| \\
&= (1 + \frac{\varepsilon}{3}) \cdot |uv| - |xu| \\
&\ge (1 + \frac{\varepsilon}{3})(1 - \frac{4}{s}) \cdot |xy| - \frac{2}{s} \cdot |xy| \\
&= (1 + \frac{\varepsilon}{3} - \frac{4\varepsilon}{3s} - \frac{6}{s}) \cdot |xy| \\
&\ge |xy|.
\end{aligned}$$

The last inequality follows from $s = 8 + 18/\varepsilon \ge 4 + 18/\varepsilon$.

(ii): Let q_x be the point on the perimeter of $D(u, (1 + \frac{2}{3}\varepsilon)|uv|)$ furthest from x. It suffices to show that $|xq_x| \le (1 + \varepsilon) \cdot |xy|$. We use the triangle

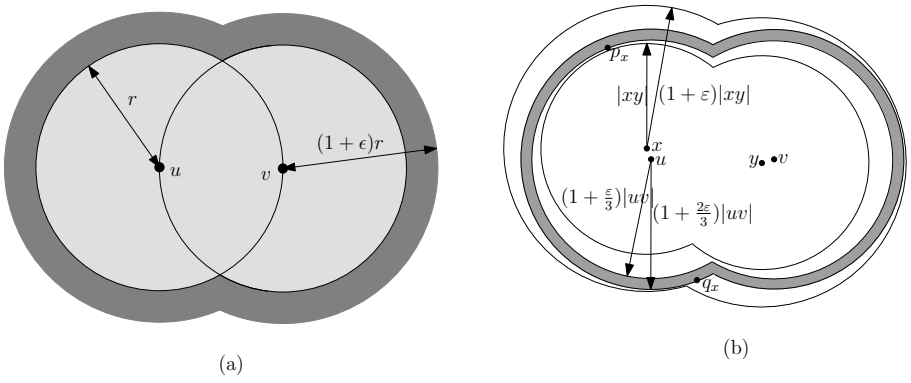

Fig. 2: (a) The light gray area represents the (exact) interference region, that is, the sphere $S(u, v)$, while the larger region represents the estimated interference region, that is, the sphere $S_{1+\varepsilon}(u, v)$. (b) Illustration for the proof of Lemma 6.

inequality and Lemma 5(ii) and (iii).

$$\begin{aligned} |xq_x| &\leq |q_x u| + |xu| \\ &= (1 + \frac{2\varepsilon}{3}) \cdot |uv| + |xu| \\ &\leq (1 + \frac{2\varepsilon}{3})(1 + \frac{4}{s}) \cdot |xy| + \frac{2}{s} \cdot |xy| \\ &= (1 + \frac{2\varepsilon}{3} + \frac{8\varepsilon}{3s} + \frac{6}{s}) \cdot |xy| \\ &\leq (1 + \varepsilon) \cdot |xy|. \end{aligned}$$

The last inequality follows from the assumption that $s \geq 8 + 18/\varepsilon$. □

Next we define the edge set $\widetilde{E}^{\varepsilon}$ as follows. For each well-separated pair $\{A_i, B_i\}$ select exactly one pair of points u, v such that $u \in A_i$ and $v \in V_i$. The edge $\{u, v\}$ is added to $\widetilde{E}^{\varepsilon}$ setting its weight to be the ε-valid estimation $\mathrm{Int}_{\varepsilon}(\{u, v\})$. Note that the number of edges in E^{ε} is $n(n-1)/2$ while the number of edges in $\widetilde{E}^{\varepsilon}$ is bounded by $O(n/\varepsilon^2)$, that is, the number of well-separated pairs.

3.3 Computing minimum-estimated-interference networks

We will use the same approach as we used in Section 2, with one difference, instead of using the graphs G_j we will use the graphs G_j^{ε}. Actually we will go one step further, instead of using G_j^{ε} we will use $\widetilde{G}_j^{\varepsilon} = (V, \widetilde{E}_j^{\varepsilon})$.

The general algorithm for finding a minimum-interference network based on estimated interferences is given in Algorithm 2. Next we analyze the running time of the general algorithm and then prove the correctness for each of the three properties that we consider.

Lemma 7 *The graph $\widetilde{G}^{\varepsilon} = (V, \widetilde{E}^{\varepsilon})$ can be constructed in $O(n/\varepsilon^2(\log n + 1/\varepsilon))$ time.*

Algorithm 2 MinEstimatedInterferenceNetwork$(V, \mathcal{P})$

Compute $\widetilde{G}^\varepsilon \leftarrow (V, \widetilde{E}^\varepsilon)$
$upper \leftarrow 1/4$
repeat
  $upper \leftarrow 2 \cdot upper$
  $\widetilde{E}^\varepsilon_{upper} \leftarrow$ all edges in $\widetilde{E}^\varepsilon$ with weight at most $upper$
  $\widetilde{G}^\varepsilon_{upper} \leftarrow (V, \widetilde{E}^\varepsilon_{upper})$
until *FulfillsProperty*$(\widetilde{G}^\varepsilon_{upper}, \mathcal{P})$
$lower \leftarrow \lfloor upper/2 \rfloor$
while $upper > lower + 1$ **do**
  $middle \leftarrow \frac{1}{2}(lower + upper)$
  $\widetilde{E}^\varepsilon_{middle} \leftarrow$all edges in $\widetilde{E}^\varepsilon$ with weight at most $middle$
  $\widetilde{G}^\varepsilon_{middle} \leftarrow (V, \widetilde{E}^\varepsilon_{middle})$
  if *FulfillsProperty*$(\widetilde{G}^\varepsilon_{middle}, \mathcal{P})$ **then** $upper \leftarrow middle$
  else $lower \leftarrow middle$
return $\widetilde{G}^\varepsilon_{upper}$

Proof. Initially set $\widetilde{E}^\varepsilon$ to be empty. Construct a well-separated pair decomposition with respect to $s = 8 + 18/\varepsilon$ in $O(n/\varepsilon^2 + n \log n)$ time. For each well-separated pair $\{A_i, B_i\}$ select an arbitrary pair of points $u \in A_i$ and $v \in B_i$. Perform an ε'-approximate range counting query [AM00] with $S_{1+\varepsilon/2}(u, v)$ as the query range, where $\varepsilon' = \frac{\varepsilon}{12+6\varepsilon}$. The number of reported points (minus u and v) determines an estimated interference between $|V \cap S_{1+\varepsilon/3}(u, v)|$ and $|V \cap S_{1+2\varepsilon/3}(u, v)|$ and hence an ε-estimated interference for every edge in E^ε, according to Lemma 6. Since the query range has constant complexity, each such query can be answered in $O(\log n + 1/\varepsilon)$ time with the BBD-tree by Arya and Mount [AM00] (following the analysis by Haverkort et al. [HdBG04]). In total this requires $O(n/\varepsilon^2(\log n + 1/\varepsilon))$ time since $O(n/\varepsilon^2)$ queries are performed. □

Theorem 2 *Algorithm* MinEstimatedInterferenceNetwork *runs in* $O(n/\varepsilon^2(\log n + 1/\varepsilon) + P(n, n/\varepsilon^2) \log k)$ *time, where* n *is the number of nodes,* k *is the maximum* ε*-valid estimated interference of any edge in the network output, and* $P(n, m)$ *is the running time of FulfillsProperty on a graph with* n *nodes and* $O(m)$ *edges.*

Proof. The graph $\widetilde{G}^\varepsilon = (V, \widetilde{E}^\varepsilon)$ is constructed in $O(n/\varepsilon^2(\log n + 1/\varepsilon))$ time. The exponential and binary search is iterated at most $\lceil \log k \rceil$ times. Each time the edge set $\widetilde{E}^\varepsilon_j$ is computed in $O(n/\varepsilon^2)$ time by looking at all edges in $\widetilde{E}^\varepsilon$. The graph is tested for the desired property in $O(P(n, n/\varepsilon^2))$ time. Summing up the time bounds gives a total of $O(n/\varepsilon^2(\log n + 1/\varepsilon) + P(n, n/\varepsilon^2) \log k)$ time. □

Before we study the three desirable properties (connectivity, bounded dilation, and bounded link diameter) we need two basic properties of the graph $\widetilde{G}^\varepsilon$.

3.4 Two properties of the sparse graph

For technical reasons we have to ensure that the interference estimations of all shortest edges in G^ε are zero. This can be accomplished during the processing of the well-separated pair decomposition without requiring any extra time.

In Theorem 3 below we show that $\widetilde{G}_j^\varepsilon$ closely approximates G_j^ε. For the proof we require the following technical lemma.

Lemma 8 *For a well-separated pair $\{A_i, B_i\}$ let $\{u, v\}$ be an arbitrary edge, where $u \in A_i$ and $v \in B_i$, with ε-valid interference estimation j. Then for every edge $\{x, y\}$ with $x, y \in A_i$ or $x, y \in B_i$ it holds that every ε-valid interference estimation of $\{x, y\}$ is at most j.*

Proof. The distance between x and y is bounded by $2/s \cdot |uv|$ according to Lemma 5. By the choice of s this is less than $\varepsilon/9 \cdot |uv|$ and thus, for reasonably small values of ε, it holds that $S_{1+\varepsilon}(x, y) \subseteq S(u, v)$, see Figure 3a. Hence, even if $\{u, v\}$ attains its minimum possible ε-valid interference estimation j, each ε-valid interference estimation of (x, y) is at most j. □

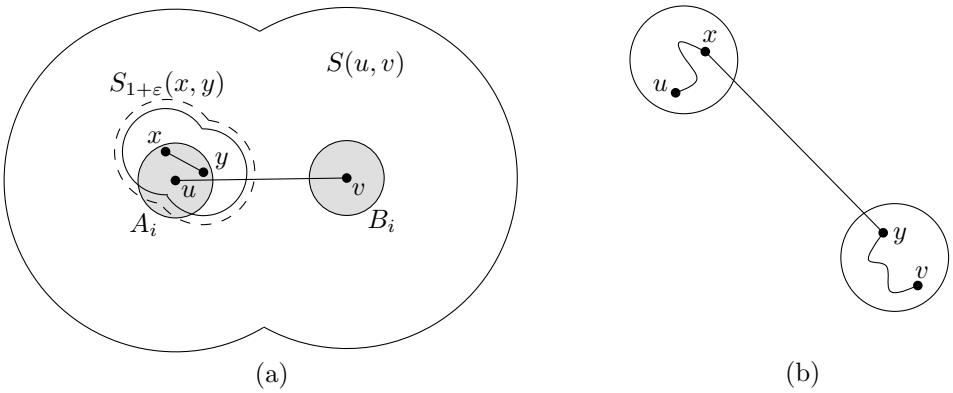

Fig. 3: (a) Illustration for the proof of Lemma 8. (b) Illustration for the proof of Theorem 3.

Let $d_j(u, v)$ and $\widetilde{d}_j(u, v)$ denote the Euclidean length of the shortest path between u and v in G_j^ε and $\widetilde{G}_j^\varepsilon$, respectively.

Theorem 3 *For any $u, v \in V$ and any non-negative integer j we have $d_j(u, v) \leq \widetilde{d}_j(u, v) \leq (1 + \varepsilon) \cdot d_j(u, v)$.*

Proof. Since $\widetilde{E}_j^\varepsilon$ is a subset of E_j^ε the left inequality follows. It remains to prove the right inequality. For this it is sufficient to show that for every edge $e = \{u, v\}$ in E_j^ε there is a path P in $\widetilde{E}_j^\varepsilon$ such that $|uv| \leq |P| \leq (1 + \varepsilon)|uv|$.

Sort the edges of E_j^ε with respect to their length in increasing order. We denote the edges in the sorted sequence by $e_1, \ldots, e_m$. The theorem is now proven by induction on the length of the edges.

Base cases: Consider a shortest edge $e_1 = \{u, v\}$ in E_j^ε. Our claim is that e_1 is also in $\widetilde{E}_j^\varepsilon$. From Definition 4 it follows that there is a well-separated pair

$\{A_i, B_i\}$ with $A_i = \{u\}$ and $B_i = \{v\}$ (or $A_i = \{v\}$ and $B_i = \{u\}$). As $\{u, v\}$ is a closest pair in V, its interference estimation is zero by construction. Hence, e_1 is in $\widetilde{E}_j^\varepsilon$.
Induction hypothesis: Assume that the theorem holds for all edges of E_j^ε shorter than e_i.
Induction step: Consider the edge $e_i = \{u, v\}$, and let $\{A_i, B_i\}$ be the well-separated pair such that $u \in A_i$ and $v \in B_i$. According to the construction of $\widetilde{E}_j^\varepsilon$ there exists an edge $\{x, y\}$ in $\widetilde{E}_j^\varepsilon$ such that $x \in A_i$ and $y \in B_i$, see Figure 3b. For ease of explanation we assume that $\{x, y\} \cap \{u, v\} = \emptyset$ (the case of $\{x, y\} \cap \{u, v\} \neq \emptyset$ is only easier, as the rest of the proof will show).

The length of $\{x, y\}$ is at most $(1 + 4/s) \cdot |uv|$, according to Lemma 5. According to Lemma 8, the edges $\{u, x\}$ and $\{y, v\}$ are in E_j^ε. Hence, as $\{u, x\}$ and $\{y, v\}$ are shorter than e_i, there is path P_{ux} between u and x of length at most $(1+\varepsilon)\cdot|ux|$, and a path P_{yv} between y and v of length at most $(1+\varepsilon)\cdot|yv|$ in $\widetilde{G}_j^\varepsilon$ by induction hypothesis. For the u-v path $P_{ux}, \{x, y\}, P_{yv}$ in $\widetilde{G}_j^\varepsilon$ we have:

$$\begin{aligned}
\widetilde{d}(u, v) &\leq \widetilde{d}(u, x) + \widetilde{d}(x, y) + \widetilde{d}(y, v) \\
&\leq (1 + \varepsilon) \cdot |ux| + \left(1 + \frac{4}{s}\right) \cdot |uv| + (1 + \varepsilon) \cdot |yv| \\
&\leq (1 + \varepsilon) \cdot \frac{2}{s}|uv| + \left(1 + \frac{4}{s}\right) \cdot |uv| + (1 + \varepsilon) \cdot \frac{2}{s}|uv| \\
&= \left(1 + \frac{8}{s} + \frac{4\varepsilon}{s}\right) \cdot |uv| \\
&\leq (1 + \varepsilon) \cdot |uv|.
\end{aligned}$$

In the third inequality we used Lemma 5 and in the final inequality we used that s was chosen to be $8 + 18/\varepsilon \geq 4 + 8/\varepsilon$. □

3.5 Minimum-estimated interference spanning trees

We start with the most basic property, namely connectivity. We will prove the corresponding estimated versions of Corollaries 1 and 2 in Section 2.3.

From Theorem 3 it immediately follows that G_j^ε is connected if and only if $\widetilde{G}_j^\varepsilon$ is connected. Thus a simple approach to test the connectivity is to use a breadth-first search in $\widetilde{G}_j^\varepsilon$ which takes linear time with respect to the size of $\widetilde{E}_j^\varepsilon$. By filling in $P(n, kn) = O(n/\varepsilon^2)$ in Theorem 2 we get:

Corollary 5 *We can compute a minimum-estimated-interference connected graph in $O(n/\varepsilon^2(\log n + 1/\varepsilon))$ time.*

And by running Kruskal's algorithm on the graph $\widetilde{G}_k^\varepsilon$ we get:

Theorem 4 *We can compute a minimum-estimated-interference spanning tree $\mathcal{T}$ of V with $\sum_{e\in\mathcal{T}} \mathrm{Int}_\varepsilon(e) = \sum_{e\in\mathcal{T}_{\min}} \mathrm{Int}_\varepsilon(e)$ in $O(n/\varepsilon^2(\log n + 1/\varepsilon))$ time, where $\mathcal{T}_{\min}$ is a minimum spanning tree of G_k^ε with respect to Int_ε.*

Proof. Running Kruskal's algorithm [Kru56] on $\widetilde{G}_k^\varepsilon$ takes $O(n/\varepsilon^2 + n \log n)$ time and thus the running time is dominated by the computation of $\widetilde{G}_k^\varepsilon$ according to Corollary 5.

It remains to show that $\widetilde{G}_k^\varepsilon$ contains a spanning tree $\mathcal{T}$ of cost equal to the cost of $\mathcal{T}_{\min}$. Consider the edges in E_k^ε and order them with respect to their interference estimations. If two edges have the same interference estimation then we order them with respect to the Euclidean distance between their endpoints. For simplicity, let $\mathcal{T}_{\min}$ denote the minimum spanning tree obtained by running Kruskal's algorithm using this ordering. Let $\{u_1, v_1\}, \{u_2, v_2\}, \ldots, \{u_{n-1}, v_{n-1}\}$ be the edges of $\mathcal{T}_{\min}$ in the order in which they were added to $\mathcal{T}_{\min}$.

Consider the subgraph $\widetilde{F}$ of $\widetilde{G}_k^\varepsilon$ constructed as follows: for each edge $\{u_j, v_j\}$ in $\mathcal{T}_{\min}$, find the well-separated pair (A_i, B_i) such that $u_j \in A_i$ and $v_j \in B_i$, and add the edge $\{x, y\} \in \widetilde{G}_k^\varepsilon$ with $x \in A_i$ and $y \in B_i$ to $\widetilde{F}$. Clearly the cost of $\widetilde{F}$ is equal to the cost of $\mathcal{T}_{\min}$, since for every edge $\{u, v\}$ in $\mathcal{T}_{\min}$ an edge $\{x, y\}$ is added to $\widetilde{F}$ whose interference estimation equals the estimation of $\{u, v\}$. It remains to prove that $\widetilde{F}$ is a spanning tree.

We will prove the following statement:

For every edge $\{u_j, v_j\}$ in $\mathcal{T}_{\min}$ there is a path in $\widetilde{F}$ between u_j and v_j. (*)

Since $\widetilde{F}$ contains $n-1$ edges, it follows that $\widetilde{F}$ must be a spanning tree. Our proof of (*) is by induction over the index j of the edges in $\mathcal{T}_{\min}$.

Base case: The first edge added to $\mathcal{T}_{\min}$ is $\{u_1, v_1\}$. By construction, $\{u_1, v_1\}$ is the shortest edge in $\binom{V}{2}$ and its interference estimation is zero. Then the well-separated pair that contains it must be a pair (A_i, B_i) with $A_i = \{u_1\}$ and $B_i = \{v_1\}$ and hence, $\{u_1, v_1\}$ is also in $\widetilde{F}$.

Induction hypothesis: Assume that condition (*) holds for all edges $\{u_1, v_1\}, \ldots, \{u_{i-1}, v_{i-1}\}$ in $\mathcal{T}_{\min}$.

Induction step: Let $\{x_i, y_i\}$ be the edge in $\widetilde{F}$ corresponding to $\{u_i, v_i\}$, and let $\{A_i, B_i\}$ be the well-separated pair such that $u_i, x_i \in A_i$ and $v_i, y_i \in B_i$. Our claim is that there is a path between u_i and x_i, and a path between v_i and y_i, only containing edges from the set $\{\{u_1, v_1\}, \ldots, \{u_{i-1}, v_{i-1}\}\}$. According to Lemma 8 an interference estimation of the edge $\{u_i, x_i\}$ does not exceed the interference estimation of the edge $\{u_i, v_i\}$, and obviously $\{u_i, x_i\}$ is shorter than $\{u_i, v_i\}$. Thus $\{u_i, x_i\}$ was considered before $\{u_i, v_i\}$ and either it was added to $\mathcal{T}_{\min}$, or u_i and x_i were already connected in $\mathcal{T}_{\min}$ and adding $\{u_i, x_i\}$ would have created a cycle in $\mathcal{T}_{\min}$ (at the time when $\{u_i, x_i\}$ would have been inserted according to the order). In both cases it follows that u_i and x_i are connected in $\mathcal{T}_{\min}$ by a path that only contains edges from the set $\{\{u_1, v_1\}, \ldots, \{u_{i-1}, v_{i-1}\}\}$, and thus, by the induction hypothesis, they are also connected in $\widetilde{F}$. The same argument yields a path in $\widetilde{F}$ between v_i and y_i. This completes the proof of (*). □

3.6 Minimum-estimated-interference t-spanners

We use the same approach as in Section 2.4 and apply Dijkstra's algorithm to compute the dilation in $\widetilde{G}_j^\varepsilon$. This takes $O(n^2/\varepsilon^2 + n^2 \log n)$ time. Note that according to Theorem 3, $\widetilde{G}_j^\varepsilon$ is at least an $(1+\varepsilon)t$-spanner if G_j^ε is a t-spanner. Thus, we implement the answer of the routine *FulfillsProperty* to be 'yes' if and only if $\widetilde{G}_j^\varepsilon$ is an $(1+\varepsilon)t$-spanner. This only implies that G_j^ε is an $(1+\varepsilon)t$-spanner, but the interference of G_j^ε is at most the minimum-estimated interference of a t-spanner with respect to Int_ε. Thus, $P(n, n/\varepsilon^2) = O(n^2/\varepsilon^2 + n^2 \log n)$ and by applying Theorem 2 we obtain:

Corollary 6 *In $O(n^2 \log k(1/\varepsilon^2 + \log n))$ time we can compute a $(1+\varepsilon)t$-spanner of V with estimated interference at most the minimum-estimated interference of any t-spanner of V.*

3.7 Minimum-estimated-interference d-hop networks

When one wants to construct a d-hop network with minimum estimated interference value it seems hard to avoid studying all the edges in E_j^ε. A simple way to check if a graph is a d-hop network is to perform a breadth-first search (BFS) in G_j^ε from each of the nodes. The problem is that the running time is linear in the number of edges, thus running the n searches in G_j^ε would give a cubic running time. Also, we cannot perform the BFS in $\widetilde{G}_j^\varepsilon$ since $\widetilde{G}_j^\varepsilon$ does not approximate G_j^ε when it comes to the number of hops. We will show, however, that for every G_j^ε we can perform an implicit breadth-first search in a graph $\mathcal{T}$ that has only $O(n/\varepsilon^2)$ edges.

The graph $\mathcal{T}$ is the split tree [CK95] $\mathcal{S}$ that is built in the first step of Algorithm 2 in order to compute the WSPD, augmented with a number of non-tree edges. In particular, $\mathcal{S}$ is a binary tree in which each vertex corresponds to a subset of V. We identify each vertex with its corresponding point set. The root of $\mathcal{S}$ is V. For any non-leaf vertex $\mathcal{A} \in \mathcal{S}$ and its two children $\mathcal{A}_1, \mathcal{A}_2$ it holds that $\mathcal{A}_1 \cap \mathcal{A}_2 = \emptyset$ and $\mathcal{A}_1 \cup \mathcal{A}_2 = \mathcal{A}$. All leaves correspond to single points and for every well-separated pair $\{A_i, B_i\}_{1 \le i \le m}$ there is exactly one vertex $\mathcal{A} \in \mathcal{S}$ with $\mathcal{A} = A_i$ and one vertex $\mathcal{B} \in \mathcal{S}$ with $\mathcal{B} = B_i$. (However, not all vertices in $\mathcal{S}$ correspond to a set A_i or B_i.) To avoid confusion we will always refer to the nodes of $\mathcal{T}$ as vertices. For a given vertex $\mathcal{A}$, our algorithm needs to access all vertices $\mathcal{B}$ such that $\{\mathcal{A}, \mathcal{B}\}$ corresponds to a pair $\{A_i, B_i\}$ of the WSPD. Therefore we extend $\mathcal{S}$ with edges between all pairs of vertices that correspond to pairs of the WSPD. This extended version of $\mathcal{S}$ constitutes the graph $\mathcal{T}$ that we will use for our breadth-first search. The non-tree edges are distinguishable from the tree edges and can be inserted without additional cost when computing the WSPD. Observe that the number of tree edges is at most $2(n-1)$ while the number of edges that join well-separated sets is $O(n/\varepsilon^2)$. Therefore $\mathcal{T}$ has $O(n/\varepsilon^2)$ edges.

The idea is now to perform n implicit BFS's, one for each point $x \in V$. The value MaxHops(x) obtained by the implicit BFS for x is the depth of a BFS-tree of G_j^ε with root x. After performing all implicit BFS's, we have access to the link diameter of G_j^ε which is $\max_{x \in V} \text{MaxHops}(x)$.

We will first give the algorithm to compute MaxHops(x) and then prove its correctness. The algorithm is essentially the same as the one by Gudmunsson et al. [GNS03] but slightly modified to fit our setting. It computes a breadth-first forest consisting of breadth-first trees rooted at all vertices of $\mathcal{T}$ which contain x, these are exactly x itself and all its ancestors in the split tree. As usually the breadth-first search features a queue Q of vertices that still have to be processed and it terminates as soon as Q is empty. For each vertex $\mathcal{A}$ of the split tree, the algorithm maintains two variables:

- *color*($\mathcal{A}$), whose value is either *white*, *gray*, or *black*, and
- *dist*($\mathcal{A}$), whose value is the minimum distance to get from x to at least one point of $\mathcal{A}$ in G_j^ε (as computed by the algorithm).

The algorithm will maintain the following invariants: All vertices that are white have not been visited yet. The vertices that are grey have been visited and are stored in the queue Q. All vertices $\mathcal{A}$ that are black have been processed already and $dist(\mathcal{A})$ contains the correct distance from x.

Step 1: For each vertex $\mathcal{A} \in \mathcal{T}$ set $color(\mathcal{A}) := white$ and $dist(\mathcal{A}) := \infty$.

Step 2: Initialize an empty queue Q. Starting at the leaf of $\mathcal{T}$ storing x, walk up the tree to the root. For each vertex $\mathcal{A}$ encountered, set $color(\mathcal{A}) := gray$ and, if $\mathcal{A}$ corresponds to a set in a well-separated pair, set $dist(\mathcal{A}) := 0$ and append $\mathcal{A}$ to the end of Q.

Step 3: If Q is empty then go to Step 4. Otherwise, let $\mathcal{A}$ be the first element of Q. Delete $\mathcal{A}$ from Q and set $color(\mathcal{A}) := black$. For each well-separated pair $\{A_i, B_i\}$ for which $A_i = \mathcal{A}$ (or $B_i = \mathcal{A}$) and whose edges have interference estimations of at most j, let $\mathcal{B}$ be the vertex in $\mathcal{T}$ with $\mathcal{B} = B_i$ ($\mathcal{B} = A_i$). If $\mathcal{B}$ is white do the following:

Step 3.1: Starting at vertex $\mathcal{B}$, walk up the split tree until the first non-white vertex is reached. For each white vertex $\mathcal{A}'$ encountered, set $color(\mathcal{A}') := gray$ and, if $\mathcal{A}'$ corresponds to a set in a well-separated pair, set $dist(\mathcal{A}') := dist(\mathcal{A})+1$ and add $\mathcal{A}'$ to the end of Q.

Step 3.2: Visit all vertices in the subtree of $\mathcal{B}$. For each vertex $\mathcal{A}'$ in this subtree, set $color(\mathcal{A}') := gray$ and, if $\mathcal{A}'$ corresponds to a well-separated set, set $dist(\mathcal{A}') := dist(\mathcal{A}) + 1$ and add $\mathcal{A}'$ to the end of Q.

After all such vertices $\mathcal{B}$ have been processed, go to Step 3.

Step 4: Return $\text{MaxHops}(x) = \max\{\text{dist}(\mathcal{A}) \mid \mathcal{A} \in \mathcal{T} \text{ is a set in a well-separated pair}\}$.

Observe that, if $\mathcal{A}'$ is the first non-white vertex reached in Step 3.1, all vertices on the path from $\mathcal{A}'$ to the root of the split tree are non-white. Also, if $color(\mathcal{B}) = white$, then all vertices in the subtree of $\mathcal{B}$ (these are visited in Step 3.2) are white.

To estimate the running time of the algorithm, we first note that Step 1 as well as Step 2 takes $O(n)$ time. The total time for Step 3 is proportional to the number of edges in $\mathcal{T}$ and the total time for walking through $\mathcal{T}$ in Steps 3.1 and 3.2. It follows from the algorithm that each edge of $\mathcal{T}$ is traversed at most once. Therefore, Step 3 takes $O(n/\varepsilon^2)$ time, which is also the total running time of the BFS from x. We now prove its correctness.

Theorem 5 *The value returned by the above algorithm is* $\text{MaxHops}(x)$.

Proof. We show (*): for each $\mathcal{A} \in \mathcal{T}$ that corresponds to a set in a well-separated pair, $dist(\mathcal{A})$ is the minimum distance to get from x to at least one point of $\mathcal{A}$ in G_j^ε. This proves the claim as all nodes V are stored in a leaf of $\mathcal{T}$ and correspond to a well-separated set, thus the set in Step 4 over which the maximum is taken contains all the distances from x to every other node in G_j^ε and no bigger values than that.

Note that, besides initializing with ∞, $dist(\mathcal{A})$ is set only once for each $\mathcal{A}$, namely when $\mathcal{A}$ is encountered for the first time. We show the following two invariants: (i) when $dist(\mathcal{A})$ is set, there is at least one node in $\mathcal{A}$ that is at most

$dist(\mathcal{A})$ steps away from x in G_j^ε and (ii) no vertex in $\mathcal{A}$ can be reached from x by less than $dist(\mathcal{A})$ steps. This proves (*). Step 2 sets $dist(\mathcal{A})$ to zero for all $\mathcal{A} \in \mathcal{T}$ that contain x and thus fulfills the two invariants for these vertices.

Next, we show that invariant (i) holds for Step 3.1 and Step 3.2. Let $\{A_i, B_i\}$ be the well-separated pair that caused the calls of Step 3.1 and Step 3.2 and let $\mathcal{B} = B_i$. Since $dist(A_i) = dist(B_i) - 1$, we can prove by induction that the invariant holds for A_i and thus there is at least one point $a \in A_i$ that can be reached from x in $dist(\mathcal{B}) - 1$ steps. However, as the interference estimations of all edges between A_i and B_i is at most j this means that in G_j^ε all points in $\mathcal{B}$ can be reached from x via a in $dist(\mathcal{B})$ steps. This shows invariant (i) for Step 3.1 as all vertices on the path from $\mathcal{B}$ to the first non-white ancestor contain supersets of the point set that corresponds to $\mathcal{B}$. It also shows invariant (i) for Step 3.2 as the vertices in the subtree of $\mathcal{B}$ contain subsets of the point set that corresponds to $\mathcal{B}$.

We prove that invariant (ii) holds throughout the algorithm by contradiction. Let $\mathcal{B} \in \mathcal{T}$ be the first vertex encountered for which (ii) is violated. This means that there exists a path $e_1, \ldots, e_\ell \in G_j^\varepsilon$ connecting x and a node $v \in \mathcal{B}$ with $\ell < dist(\mathcal{B})$ edges. Let $e_\ell = \{u, v\}$ and $\{A_i, B_i\}$ be the unique well-separated pair with $u \in A_i$ and $v \in B_i$. The path $e_1, \ldots, e_{\ell-1}$ connects x with $u \in A_i$ by $\ell - 1$ edges. Since $\mathcal{B}$ is a minimal counterexample, invariant (ii) holds for A_i which means that $dist(A_i) \leq \ell - 1$. This is a contradiction because then $dist(\mathcal{B}) \leq dist(A_i) + 1 = \ell$ as again all edges between A_i and B_i are in G_j^ε. □

Recall that we can implement the function *FulfillsProperty*(G_j^ε, *"has link-diameter at most d"*) as follows. We augment the split tree $\mathcal{S}$ of the WSPD with the $O(n/\varepsilon^2)$ edges that correspond to well-separated sets whose representative edges have interference estimation of at most j. Then we run the above algorithm once for each node in V. We have $P(n, n/\varepsilon^2) = O(n^2/\varepsilon^2)$ and by applying Theorem 2 we get:

Corollary 7 *Given a set V of n points in the plane and an integer d, one can compute a d-hop network with minimum estimated interference value in $O(n/\varepsilon^2(n \log k + 1/\varepsilon))$ time.*

4 Generalizations and extensions

In this section we consider different models and different interference functions, and we discuss the possibility to modify the presented algorithms to handle these cases.

Bounded transmission radius. Burkhart et al. [BvRWZ04] and Moaveni-Nejad and Li [ML05] considered the case where each transmitter has a bounded transmission radius. The algorithms in Section 2 can be modified to handle this case. In the estimated interference model a fundamental result breaks down, namely Lemma 8. However, this result, and all other results in Section 3, can easily be shown to hold if the input is guaranteed to have the following (reasonable) property:

For every point p with transmission radius $T(p)$ it holds that every point within distance $\varepsilon \cdot T(p)$ of p has transmission radius at least $2\varepsilon \cdot T(p)$, for some given positive constant ε.

Weighted points. In the scenario in which the points are weighted, the weights can be thought of as a measure of importance assigned to the nodes. Points with a very high weight are very important and thus interference with those nodes should be avoided. The changes to the algorithms in Section 2 are trivial, since range counting queries can easily be modified to handle weighted points. In the estimated interference model the same bounds also hold for the weighted case. The only difference is in computing the interference value of an edge – fortunately the BBD-tree [AM00] can also be used for weighted range queries without additional preprocessing or query time.

References

[AM00] Sunil Arya and David M. Mount. Approximate range searching. *Comput. Geom. Theory Appl.*, 17:135–152, 2000. 13, 20

[BvRWZ04] Martin Burkhart, Pascal von Rickenbach, Roger Wattenhofer, and Aaron Zollinger. Does topology control reduce interference? In *Proc. 5th ACM Int. Symp. Mobile Ad Hoc Networking and Computing (MobiHoc'04)*, pages 9–19, 2004. 2, 3, 5, 9, 19

[Cha00] Timothy M. Chan. Random sampling, halfspace range reporting, and construction of ($\leq k$)-levels in three dimensions. *SIAM J. Comput.*, 30:561–575, 2000. 7

[CK95] Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42(1):67–90, January 1995. 10, 11, 17

[Dij59] Edsger W. Dijkstra. A note on two problems in connection with graphs. *Numerische Math.*, 1:269–271, 1959. 9

[DN97] Gautam Das and Giri Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *Internat. J. Comput. Geom. Appl.*, 7:297–315, 1997. 9

[GHvK02] Joachim Gudmundsson, Mikael Hammar, and Marc van Kreveld. Higher order delaunay triangulations. *Comput. Geom. Theory Appl.*, 23(1):85–98, 2002. 6

[GLN02] Joachim Gudmundsson, Christos Levcopoulos, and Giri Narasimhan. Improved greedy algorithms for constructing sparse geometric spanners. *SIAM J. Comput.*, 31(5):1479–1500, 2002. 9

[GNS03] Joachim Gudmundsson, Giri Narasimhan, and Michiel Smid. Distance-preserving approximations of polygonal paths. In *Proc. 23rd Conf. Foundations Software Technology Theoretical Comput. Sci. (FSTTCS'03)*, pages 217–228, 2003. 17

[HdBG04] Herman Haverkort, Mark de Berg, and Joachim Gudmundsson. Box-trees for collision checking in industrial installations. *Comput. Geom. Theory Appl.*, 28(2–3):113–135, 2004. 13

[JT92] Jerzy W. Jaromczyk and Godfried T. Toussaint. Relative neighborhood graphs and their relatives. *Proc. IEEE*, 80(9):1502–1517, 1992. 5

[Kru56] Joseph B. Kruskal. On the shortest spanning subtree of a graph and the travelling salesman problem. *Proc. American Mathematical Society*, 7:48–50, 1956. 15

[Lee82] Der-Tsai Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Trans. Comput.*, C-31:478–487, 1982. 7

[LW04] Xiang-Yang Li and Yu Wang. Minimum power assignment in wireless ad hoc networks with spanner property. In *Proc. IEEE Workshop on High Performance Switching and Routing (HPSR'04)*, pages 231–235, 2004. 9

[Mat93] Jiří Matoušek. Range searching with efficient hierarchical cuttings. *Discrete Comput. Geom.*, 10(2):157–182, 1993. 7

[ML05] Kousha Moaveni-Nejad and Xiang-Yang Li. Low-interference topology control for wireless ad hoc networks. *Internat. J. Ad Hoc & Sensor Wireless Networks*, 1(1–2):41–64, 2005. 2, 3, 4, 19

[MT87] Alistair Moffat and Tadao Takaoka. An all pairs shortest path algorithm with expected time $O(n^2 \log n)$. *SIAM J. Comput.*, 16(6):1023–1031, 1987. 10

[NS07] Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007. 9

[Pra99] Ravi Prakash. Unidirectional links prove costly in wireless ad-hoc networks. In *Proc. 3rd Int. Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIALM'99)*, pages 15–22. ACM Press, 1999. 2

[Pri57] R. C. Prim. Shortest connection networks and some generalisations. *Bell Systems Technical Journal*, pages 1389–1410, 1957. 3

[Ram99] Edgar A. Ramos. On range reporting, ray shooting and k-level construction. In *Proc. 15th Annu. ACM Sympos. Comput. Geom. (SoCG'99)*, pages 390–399, 1999. 7

[vRSWZ05] Pascal von Rickenbach, Stefan Schmid, Roger Wattenhofer, and Aaron Zollinger. A robust interference model for wireless ad-hoc networks. In *Proc. 5th Int. Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN'05)*, 2005. CD-ROM. 2, 3