Discrete Comput Geom (2011) 45: 723–736
DOI 10.1007/s00454-011-9343-y

Kinetic Spanners in $\mathbb{R}^d$

Mohammad Ali Abam · Mark de Berg

Received: 10 July 2009 / Revised: 13 October 2009 / Accepted: 16 October 2009 /
Published online: 5 April 2011

Abstract We present a new $(1+\varepsilon)$-spanner for sets of n points in $\mathbb{R}^d$. Our spanner has size $O(n/\varepsilon^{d-1})$ and maximum degree $O(\log^d n)$. The main advantage of our spanner is that it can be maintained efficiently as the points move: Assuming that the trajectories of the points can be described by bounded-degree polynomials, the number of topological changes to the spanner is $O(n^2/\varepsilon^{d-1})$, and using a supporting data structure of size $O(n\log^d n)$, we can handle events in time $O(\log^{d+1} n)$. Moreover, the spanner can be updated in time $O(\log n)$ if the flight plan of a point changes. This is the first kinetic spanner for points in $\mathbb{R}^d$ whose performance does not depend on the spread of the point set.

Keywords Geometric spanners · Kinetic data structures

1 Introduction

Background Let $\mathcal{G} = (V, E)$ be an edge-weighted graph, and let $d_{\mathcal{G}}(u, v)$ denote the distance in $\mathcal{G}$—that is, the length of the (weighted) shortest path in $\mathcal{G}$—between u and v. Let $t \geq 1$ be a real number. A *t-spanner* of $\mathcal{G}$ is a subgraph $\mathcal{S} = (V, E_{\mathcal{S}})$ of $\mathcal{G}$ such that for any two vertices $u, v \in V$, we have $d_{\mathcal{S}}(u, v) \leq t \cdot d_{\mathcal{G}}(u, v)$. In other

M.A. was supported by MADALGO Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.
M.dB. was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no 639.023.301.

M.A. Abam (✉)
Department of Computer Science, TU Dortmund, 44221 Dortmund, Germany
e-mail: aabaam@gmail.com

M. de Berg
Department of Computing Science, TU Eindhoven, PO Box 513, 5600 MB Eindhoven,
The Netherlands

words, the distance between two vertices in the spanner approximates the distance in the original graph up to a factor t. The smallest t for which this property holds is called the *dilation* (or: *stretch factor*) of $\mathcal{S}$. In this paper we are interested in spanners in a geometric context. Here the graph $\mathcal{G}$ is the complete Euclidean graph on a set P of n points in $\mathbb{R}^d$. A *geometric t-spanner* is then a Euclidean graph $\mathcal{S}$ on P such that $d_{\mathcal{S}}(u, v) \leq t \cdot |pq|$ for all points $p, q \in P$, where $|pq|$ denotes the Euclidean distance between p and q. Since their introduction by Chew [6] in 1986, numerous papers on geometric spanners have appeared—see, for instance, the surveys [10, 14, 22]—and there is even a book devoted to geometric spanners [19]. From now on, we limit our discussion to geometric spanners.

When constructing spanners, the main goal is to obtain a small dilation while not using too many edges. As it turns out, this is indeed always possible: for any set of n points in $\mathbb{R}^d$ and any $\varepsilon > 0$, one can construct a $(1 + \varepsilon)$-spanner that uses only $O(n/\varepsilon^{d-1})$ edges [19]. The construction can be done in $O((n/\varepsilon^{d-1}) \log^{d-1} n)$ time. In fact, one can obtain spanners with a number of additional properties, such as small weight—more precisely, weight proportional to the weight of a minimum spanning tree (MST)—and bounded degree.

Most algorithms for constructing geometric spanners work on a given, fixed point set P. In some applications, however, one may want to insert and/or delete points from P. Then one would prefer not to reconstruct the spanner from scratch at each update. Gao et al. [11] were the first to study dynamic spanners. They show how to maintain a $(1 + \varepsilon)$-spanner of size $O(n/\varepsilon^d)$. Unfortunately, the update time of their structure depends on $\alpha(P)$, the *spread* of the point set P. (The spread of a point set is the ratio of the maximum pairwise distance to the minimum pairwise distance; it is always $\Omega(\sqrt{n})$ but, in general, cannot not be bounded as a function of n.) More precisely, the update time they obtain is $O((\log \alpha(P))/\varepsilon^d)$. In a series of papers, Roddity [20] and Gottlieb and Roddity [12, 13] improved this result, culminating in a spanner of size $O(n/\varepsilon^{O(d)})$ with $O((\log n)/\varepsilon^{O(d)})$ update time [13].

Another interesting variant, first studied by Gao et al. [11], is where the points in P move along continuous trajectories. This setting is inspired by simulations in molecular dynamics and mobile networks, and it is the setting we study in this paper.

Previous Results Gao et al. [11] study the problem of maintaining spanners for moving point in the *kinetic-data-structures* framework introduced by Basch et al. [4]. In this framework, one wants to maintain certain *attribute*—a $(1 + \varepsilon)$-spanner for given $\varepsilon > 0$ in our case—for a set of moving objects. This is done by maintaining, besides the attribute itself, some additional information that helps to detect when (and how) the attribute needs to be updated. In particular, a set of *certificates* (which are simple geometric tests, such as one point being to the left of another point) is maintained with the property that as long as the certificates are valid, the attribute is still correct. Whenever a certificate fails—this is called an *event*—the KDS needs to be updated. To this end, the certificate failure times are stored in an event queue. Two main criteria for evaluating the quality of a kinetic data structure (KDS) are the following.

- *Efficiency*: the worst-case number of events processed by the KDS (under the assumption that the trajectories can, e.g., be described by bounded-degree polynomials). Note that not all events lead necessarily to changes in the attribute; some

events are only needed to update the supporting data structures. A KDS is called efficient when the number of events is not much more than the worst-case number of changes to the attribute.
- *Response time:* the time needed to update the KDS at each event. The goal is to achieve polylogarithmic response time.

Note that the product of these two criteria gives a bound on the total time spent in maintaining the attribute over the entire motion. Thus, any good KDS should at least have good bounds for these two criteria. The other two criteria are the *compactness* of the KDS, which is essentially the amount of storage it uses, and the *locality*, which is the maximum number of certificates involving any given object. The latter is important since whenever an object changes its motion, the failure times of the certificates it is involved in should be recomputed (and the event queue should then be updated). A more extensive discussion of KDSs can be found in one of the survey papers by Guibas [15, 16].

Observe that the attribute that we are interested in—a $(1+\varepsilon)$-spanner of the given point set—is not uniquely defined. Thus we should compare the number of events to the worst-case *minimum* number of events processed by any kinetic spanner. For any given t, a kinetic t-spanner of subquadratic size must process $\Omega(n^2)$ events in the worst case. (To see this, consider a group of $n/2$ stationary points on the x-axis and another group of $n/2$ points on a line slightly below the x-axis, such that the second group passes the stationary points from left to right.) Since we are aiming for a $(1+\varepsilon)$-spanner, we therefore need to process $\Omega(n^2)$ events in the worst case. Thus we call a kinetic $(1+\varepsilon)$-spanner *efficient* when it processes $O(n^2 \operatorname{polylog} n)$ events. The spanner of Gao et al. processes $O(n^2 \log \alpha(P))$ events. Hence, it is only efficient when the spread of the point set is polynomial. The response time is $O((\log \alpha(P))/\varepsilon^d)$, so it depends on the spread as well.[1] Recently Abam et al. [2] presented a simple and efficient kinetic spanner for points moving in the plane. Their spanner has size $O(n/\varepsilon^2)$, it processes $O(n\lambda_{s+2}(n)/\varepsilon^2)$ events, and each event can be handled in $O(1)$ time, plus $O(\log n)$ to update the event queue. Here $\lambda_{s+2}(n)$ is the maximum length on an $(n, s+2)$ Davenport–Schinzel sequence; $\lambda_{s+2}(n)$ is near-linear in n [21]. Unfortunately, their approach cannot be generalized to higher dimensions, because they use the fact that the Delaunay triangulation has a linear number of edges (for any convex distance function)—something which is no longer true in dimensions $d \geq 3$. Moreover, their kinetic spanner is not local, since a point can be involved in $O(n)$ certificates.

This leads us to the main topic of our paper: is it possible to design a kinetic $(1+\varepsilon)$-spanner for points in dimensions $d \geq 3$ that is efficient—that is, processes only a near-quadratic number of events—and has polylogarithmic response time?

Our Results We show that it is indeed possible to obtain an efficient kinetic spanner in dimensions $d \geq 3$; our construction is also valid for $d = 2$. More precisely, we

[1] One may try to kinetize the dynamic spanner of Gottlieb and Roddity [13]. The hope would be that, since it can handle insertions and deletions in $O(\log n)$ time, one can also obtain a kinetic spanner with $O(\log n)$ response time. However, we do not know how to do this efficiently—for instance, one would have to detect efficiently when the spanner needs to be updated—nor do we know what the number of events would be.

present a new kinetic $(1+\varepsilon)$-spanner of size $O(n/\varepsilon^{d-1})$ for n moving points in $\mathbb{R}^d$ (for any fixed d) that processes $O(n^2/\varepsilon^{d-1})$ events in the worst case (assuming that the points follow bounded-degree algebraic trajectories). Each event can be handled in $O(\log^{d+1} n)$ time using an auxiliary data structure of size $O((n/\varepsilon^{d-1})\log^d n)$. Moreover, each point is involved in $O(1/\varepsilon^{d-1})$ certificates, which implies that our kinetic spanner is local and a motion update can be handled in $O(\log n)$ time. Furthermore, our dependency on ε is better than for the kinetic spanner of Abam et al. [2] (and also than in the dynamic spanner of Gottlieb and Roddity [13]): our dependency for $d=2$ is $O(1/\varepsilon)$, while the dependency of Abam et al. is $O(1/\varepsilon^2)$. Moreover, our structure processes slightly fewer events.

2 The Spanner

Let P be a set of n points in $\mathbb{R}^d$, and let ε be a given positive constant. We will first present our new algorithm to construct a spanner for P, and then we will show how to maintain the spanner when the points in P move.

2.1 The Construction

The θ-Graph Our spanner construction is based on the θ-graph approach [7, 17], which works as follows. Let θ be a suitably small (depending on ε) constant. We define a *θ-cone* to be the intersection of d nonparallel half-spaces such that the angle of any two rays emanating at the cone's apex and being inside the cone is at most θ. Now let $\mathcal{C}$ be a collection of $O(1/\theta^{d-1})$ interior-disjoint θ-cones, each with their apex at the origin, that together cover $\mathbb{R}^d$. We call the cones in $\mathcal{C}$ *canonical cones*. (When $d=2$, the canonical cones can be obtained by drawing $O(1/\theta)$ rays emanating from the origin such that the angle between two consecutive rays is at most θ.) For a cone $\sigma\in\mathcal{C}$ and a point $p\in\mathbb{R}^d$, let $\sigma(p)$ denote the translated copy of σ whose apex coincides with p.

The θ-graph for P is now constructed by adding at most $|\mathcal{C}|$ edges for each point $p\in P$. Namely, for each cone $\sigma\in\mathcal{C}$, we connect p to the point $q\in P\cap\sigma(p)$ that is closest to p (with ties broken arbitrarily). This produces a $(1+\varepsilon)$-spanner if we choose θ such that $\cos\theta-\sin\theta\geq 1/(1+\varepsilon)$—see [7, 17]. Finding the point inside $\sigma(p)$ that is closest to p is costly, however, and it is also hard to maintain such a closest point in the kinetic setting. Hence, a slightly different notion of closest point is used in the θ-graph construction. To this end, we choose for each $\sigma\in\mathcal{C}$ one of its edges as its *representative edge*, and we define $\mathrm{dist}_\sigma(p,q)$ to be the distance from p to the orthogonal projection of q onto $\sigma(p)$'s representative edge. Now, instead of connecting p to a point $q\in\sigma(p)$ that minimizes the Euclidean distance to p, we connect it to a point q that minimizes $\mathrm{dist}_\sigma(p,q)$—see Fig. 1(a). From now on, whenever we speak of "a closest point to p in $\sigma(p)$," we refer to such a point q.

To prove that the θ-graph produced in this manner is a $(1+\varepsilon)$-spanner, we can use the following lemma.

Lemma 2.1 *Let $\mathcal{C}$ be a collection of θ-cones, where $\cos 2\theta-\sin 2\theta\geq 1/(1+\varepsilon)$. Let $\sigma\in\mathcal{C}$ be a cone, and let q and r be two points in $\sigma(p)$ such that* $\mathrm{dist}_\sigma(p,r)\leq$

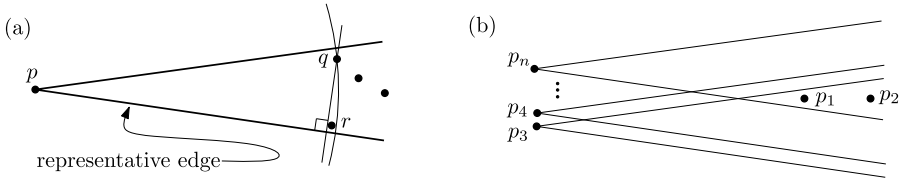

Fig. 1 (**a**) Point p will be connected to q even though r is slightly closer in terms of Euclidean distance. (**b**) The degree of p_1 in the θ-graph is $n-1$

$\mathrm{dist}_\sigma(p,q)$. *Then*

$$|pr| + (1+\varepsilon)\cdot|rq| \;\le\; (1+\varepsilon)\cdot|pq|.$$

Proof We distinguish two cases:

(i) $|pq| \ge |pr|$: Because $|pq|$ is the longest edge in the triangle pqr, we have

$$|rq| \le |pq| - (\cos\angle qpr - \sin\angle qpr)|pr|.$$

Hence,

$$\begin{aligned}|pr| + (1+\varepsilon)\cdot|rq| &\le |pr| + (1+\varepsilon)\cdot\big(|pq| - (\cos\theta - \sin\theta)|pr|\big)\\ &\le (1+\varepsilon)\cdot|pq|,\end{aligned}$$

because

$$\cos\theta - \sin\theta \ge \cos 2\theta - \sin 2\theta \ge 1/(1+\varepsilon).$$

(ii) $|pq| < |pr|$: Since $\mathrm{dist}_\sigma(p,r) \le \mathrm{dist}_\sigma(p,q)$, we have $\angle prq > \pi/2 - \angle qpr$. (See Fig. 2(a) for an illustration of this fact in the plane.) Moreover, since the segments pq and pr are inside $\sigma(p)$, we have $\angle qpr \le \theta$. Finally, from $0 \le \angle qpr \le \theta$ and $\pi/2 - \theta \le \angle prq \le \pi/2$ it follows that

$$|rq|/|pq| = \sin\angle qpr/\sin\angle prq \le \sin\theta/\cos\theta.$$

Therefore,

$$\begin{aligned}|pr| + (1+\varepsilon)\cdot|rq| \;&\le\; |pq| + |rq| + (1+\varepsilon)\cdot|rq|\\ &\le \big(1 + (2+\varepsilon)(\sin\theta/\cos\theta)\big)|pq|.\end{aligned}$$

Then to prove the lemma, we just need to show that $(1+(2+\varepsilon)(\sin\theta/\cos\theta)) \le 1+\varepsilon$ under the assumption that $\cos 2\theta - \sin 2\theta \ge 1/(1+\varepsilon)$. This inequality can simply be reduced to $(\cos\theta - \sin\theta)/(\cos\theta + \sin\theta) \ge 1/(1+\varepsilon)$. We proceed as follows:

$$\begin{aligned}(\cos\theta - \sin\theta)/(\cos\theta + \sin\theta) &= \big(\cos^2\theta - \sin^2\theta\big)/(\cos\theta + \sin\theta)^2\\ &= (\cos 2\theta)/(1 + \sin 2\theta)\\ &\ge \cos 2\theta - \sin 2\theta\\ &\ge 1/(1+\varepsilon),\end{aligned}$$

which proves our claim. □

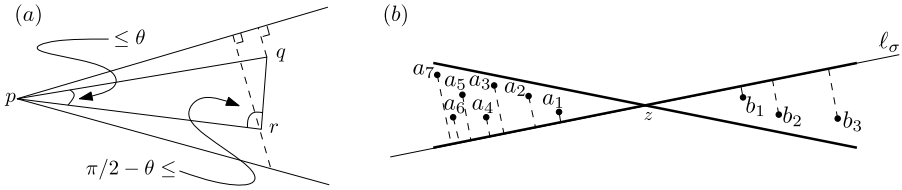

Fig. 2 (**a**) Illustration for Lemmas 2.1 and 2.4. (**b**) A_i and B_i are separated by $\sigma(z)$ and $\bar{\sigma}(z)$ for some point z. Points are labeled in order of increasing distance to z on ℓ_σ

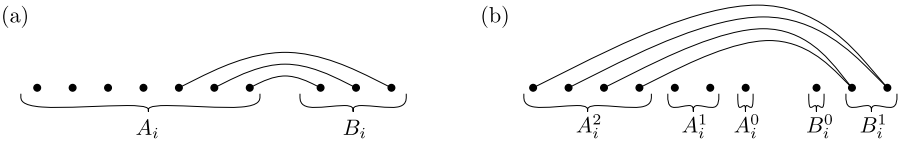

Fig. 3 (**a**) A simple method to make the pair (A_i, B_i) well-connected, which does not kinetize well. (**b**) The partitioning of A_i and B_i into groups, and one possible way of connecting A_i^2 to B_i^1 such that no point from B_i^1 receives more than two connections. Connections between other pairs of groups are not shown

Lemma 2.1 implies that concatenating the edge pr to a $(1+\varepsilon)$-path from r to q yields a $(1+\varepsilon)$-path from p to q. (A $(1+\varepsilon)$-path from p to q is a path whose length is at most $(1+\varepsilon)\cdot|pq|$.) This can be used to show by induction that if we connect every point p to the closest point in each of its cones, we get indeed a $(1+\varepsilon)$-spanner. From now on, we fix an angle θ with $\theta = \Omega(\varepsilon)$ and $\cos 2\theta - \sin 2\theta \geq 1/(1+\varepsilon)$, and we fix a set $\mathcal{C}$ of θ-cones with $|\mathcal{C}| = O(1/\theta^{d-1}) = O(1/\varepsilon^{d-1})$.

One of the disadvantages of the θ-graph is that points can get very high degree. The point p_1 in Fig. 1(b), for example, will be the closest point in $n-2$ cones shown in the figure and, hence, have degree at least $n-2$. This is especially problematic in the kinetic setting: when p_1 and p_2 in Fig. 1(b) exchange their order (in the projection onto these cone's representative edges), then we have to replace the edges $p_i p_1$ for $3 \leq i \leq n$ by the edges $p_i p_2$. Hence, we need $\Omega(n)$ time to process such an event. There are also variants of the θ-graph, such as the *ordered θ-graph* [5], where every point has low degree, but these spanners are also difficult to maintain kinetically. In the following we explain how to modify the θ-graph approach in a novel way to obtain a spanner of maximum degree $O(\log^d n)$.

The Cone-Separated Pairs Decomposition First, we introduce the concept of *cone-separated pair decomposition* (*CSPD*). Let $\bar{\sigma}(p)$ be the reflection of $\sigma(p)$ about p.

Definition 2.2 Let P be a set of points in $\mathbb{R}^d$, and let $\sigma \in \mathcal{C}$ be a cone. A *cone-separated pair decomposition*, or *CSPD* for short, for P with respect to σ is a collection $\Psi_\sigma := \{(A_1, B_1), \ldots, (A_m, B_m)\}$ of pairs of subsets from P such that

(i) For every two points $p, q \in P$ with $q \in \sigma(p)$, there is a unique pair $(A_i, B_i) \in \Psi_\sigma$ such that $p \in A_i$ and $q \in B_i$.
(ii) For any pair $(A_i, B_i) \in \Psi_\sigma$ and every two points $p \in A_i$ and $q \in B_i$, we have $q \in \sigma(p)$ and, hence, $p \in \bar{\sigma}(q)$.

By condition (ii) for any i, there must be a point $z \in \mathbb{R}^d$ such that $A_i \subset \bar{\sigma}(z)$ and $B_i \subset \sigma(z)$—see Fig. 2(b). For example, we can take z to be the apex of the intersection of all the cones centered at points in A_i. We can obtain a CSPD such that every point $p \in P$ appears in only $O(\log^d n)$ subsets in a fairly standard manner, using range-searching techniques. Next we sketch this.

Recall that the cones $\sigma(p_i)$ for $1 \le i \le n$ are all translates of the same canonical cone $\sigma \in \mathcal{C}$. Let $\mathcal{T}_\sigma$ be a (multidimensional) range tree [9] for reporting all points from P that lie in any such translated cone. Let $B(v) \subset P$ be the canonical subset of a node v at level d of the range tree, that is, the points stored at subtree rooted at v. Then we can select the points inside a query cone as the union of $O(\log^d n)$ disjoint canonical subsets $B(v)$. Moreover, any point $p \in P$ is contained in $O(\log^d n)$ canonical subsets $B(v)$. Now we perform a query with each cone $\sigma(p_i)$ for $1 \le i \le n$. For a node v at level d of the tree $\mathcal{T}_\sigma$, let $A(v)$ be the subset of points $p_i \in P$ such that $B(v)$ is one of the canonical subsets selected when we query with $\sigma(p_i)$. Our CSPD now consists of the pairs $(A(v), B(v))$. This leads to the following lemma.

Lemma 2.3 *For any set P of n points in $\mathbb{R}^d$ and any cone $\sigma \in \mathcal{C}$, there is a CSPD $\Psi_\sigma = \{(A_1, B_1), \ldots, (A_m, B_m)\}$ such that every $p \in P$ appears in $O(\log^d n)$ pairs (A_i, B_i).*

From CSPDs to Spanners Let $\mathcal{S} = (P, E_\mathcal{S})$ be a Euclidean graph on P, and let (A_i, B_i) be a pair in some CSPD Ψ_σ for P. We say that (A_i, B_i) is *well connected in* $\mathcal{S}$ if for any two points $p \in A_i$ and $q \in B_i$, we have (i) there is an edge $(p, r) \in E_\mathcal{S}$ such that $r \in \sigma(p)$ and $\mathrm{dist}_\sigma(p, r) \le \mathrm{dist}_\sigma(p, q)$, or (ii) there is an edge $(q, r) \in E_\mathcal{S}$ such that $r \in \bar{\sigma}(q)$ and $\mathrm{dist}_{\bar{\sigma}}(q, r) \le \mathrm{dist}_{\bar{\sigma}}(p, q)$.

Lemma 2.4 *Let P be a set of points, and let $\{\Psi_\sigma : \sigma \in \mathcal{C}\}$ be a collection of CSPDs for P, where $\mathcal{C}$ is a set of canonical cones as defined above. Let $\mathcal{S} = (P, E_\mathcal{S})$ be a Euclidean graph where every pair (A_i, B_i) in each Ψ_σ is well connected. Then $\mathcal{S}$ is a $(1+\varepsilon)$-spanner for P.*

Proof The proof is by induction on $|pq|$. (More precisely, we order the pairwise distances and use induction on this ordering.) Let (p, q) be the closest pair, and let $\sigma \in \mathcal{C}$ be such that $q \in \sigma(p)$. We claim that there is no point $r \in \sigma(p)$ with $\mathrm{dist}_\sigma(p, r) < \mathrm{dist}_\sigma(p, q)$. For the sake of contradiction, assume that there is such a point. Then $\angle qrp > \angle rpq$—see Fig. 2(a) and note that θ is a small angle. But this implies $|rq| < |pq|$, contradicting our assumption and proving the claim. Now let $(A_i, B_i) \in \Psi_\sigma$ be such that $p \in A_i$ and $q \in B_i$; there is such a pair, because Ψ_σ is a cone-separated pair decomposition. By the claim we just proved, p and q are adjacent in the orthogonal projection of $A_i \cup B_i$ onto the line through σ's representative edge. Since (A_i, B_i) is well connected, this implies that $(p, q) \in E_\mathcal{S}$. Therefore, the base case is true.

Now consider two arbitrary points $p, q \in P$. Let $\sigma \in \mathcal{C}$ be the cone such that $q \in \sigma(p)$, and let $(A_i, B_i) \in \Psi_\sigma$ be the pair such that $p \in A_i$ and $q \in B_i$. Since (A_i, B_i) is well connected, we can assume without loss of generality that there is a point $r \in P \cap \sigma(p)$ such that $\mathrm{dist}_\sigma(p, r) \le \mathrm{dist}_\sigma(p, q)$ and $(p, r) \in E_\mathcal{S}$. Then $|pq| >$

$|rq|$; this follows in the same way as the claim for the base case. By the induction hypothesis, there is a $(1+\varepsilon)$-path in $\mathcal{S}$ between r and q. Hence, by Lemma 2.1 there is a $(1+\varepsilon)$-path in $\mathcal{S}$ between p and q. □

It remains to show how to make the pairs $(A_i, B_i) \in \Psi_\sigma$ well connected without using too many edges. Next we explain how this can be done.

As remarked above, there must be a point $z \in \mathbb{R}^d$ such that $A_i \subset \bar{\sigma}(z)$ and $B_i \subset \sigma(z)$. Let ℓ_σ be a line through z that is parallel to the representative edge of σ—see Fig. 2(b). Project all the points in $A_i \cup B_i$ onto ℓ_σ. Then the projections of the points in A_i lie on one side of z, while the projections of the points in B_i lie on the other side of z. Label the points from A_i as $a_1, \ldots, a_k$ and label the points from B_i as $b_1, \ldots, b_l$, both in order of increasing distance to z. An easy way to make sure that (A_i, B_i) is well connected is to add the edge (a_i, b_i) to our graph for any $1 \leq i \leq \min(k, l)$, as depicted in Fig. 3. Unfortunately, such a set of edges is costly to maintain when the points move: when a new point enters a cone, we may have to change many edges. Therefore, we proceed as follows. We partition the set A_i into a logarithmic number of groups $A_i^0, \ldots, A_i^h$ such that $A_i^j := \{a_{2^j}, \ldots, a_{2^{j+1}-1}\}$. We say that a group A_i^j is *full* when it contains exactly 2^j points. Note that every group is full, except possibly for the last group, A_i^h. Similarly, we partition B_i into groups B_i^j. Next, we define the set $E(A_i, B_i)$ of edges connecting the points in A_i to those in B_i.

- For each group A_i^j, with $j > 0$, we add a collection of edges as follows. Let $k := |B_i^{j-1}|$. Consider the first $2k$ points of A_i^j, that is, the points $a_{2^j}, \ldots, a_{2^j+2k-1}$. We connect each of these $2k$ points to one of the points in B_i^{j-1} in such a way that each point in B_i^{j-1} receives only two connections.
- For each group B_i^j with $j > 0$, we add a collection of edges in the same way: we connect each of the first $2|A_i^{j-1}|$ points in B_i^j to a point in A_i^{j-1} in such a way that each point in A_i^{j-1} receives only two connections.
- We connect the point in A_i^0 to the point in B_i^0.

Lemma 2.5 *The set $E(A_i, B_i)$ of edges connects every point in A_i to at most three points in B_i, and vice versa. Moreover, the pair (A_i, B_i) is well connected by $E(A_i, B_i)$.*

Proof It follows directly from the construction that each point is connected to at most three other points, as claimed. To prove that (A_i, B_i) is well connected, consider a pair of points p, q with $p \in A_i^j$ and $q \in B_i^{j'}$. We will show that either (i) there is an edge $(p, r) \in E(A_i, B_i)$ such that $r \in \sigma(p)$ and $\mathrm{dist}_\sigma(p, r) \leq \mathrm{dist}_\sigma(p, q)$, or (ii) there is an edge $(q, r) \in E(A_i, B_i)$ such that $r \in \bar{\sigma}(q)$ and $\mathrm{dist}_{\bar{\sigma}}(q, r) \leq \mathrm{dist}_{\bar{\sigma}}(p, q)$.

Assume without loss of generality that $j \leq j'$. If $j = j' = 0$, we are done, since there is an edge connecting the point in A_i^0 (which is p) to the point in B_i^0 (which is q). If $j > 0$, then B_i^{j-1} must exist and be full. Hence, $E(A_i, B_i)$ includes an edge (p, r) for p to some $r \in B_i^{j-1}$. Since $q \in B_i^{j'}$ and $j' \geq j > j - 1$, we thus have $\mathrm{dist}_\sigma(p, r) \leq \mathrm{dist}_\sigma(p, q)$, and we are done. □

By applying the above procedure to every pair (A_i, B_i) in each Ψ_σ, we thus obtain a graph $\mathcal{S} = (P, E_\mathcal{S})$ which is a $(1+\varepsilon)$-spanner. The number of edges in $\mathcal{S}$ will be $O(n \log^d n/\varepsilon^{d-1})$, however. Next we show how to remedy this.

A Linear-Size Spanner In the following we show how to get a spanner $\mathcal{S}^* = (P, E_{\mathcal{S}^*})$ with a linear number of edges, by pruning some of the edges from $\mathcal{S} = (P, E_\mathcal{S})$. Consider an edge $(p, q) \in E_\mathcal{S}$ and a cone $\sigma \in \mathcal{C}$ such that $q \in \sigma(p)$ and $p \in \bar{\sigma}(q)$. Then we add the edge (p, q) to $E_{\mathcal{S}^*}$ if and only if (i) among all points $r \in \sigma(p)$ such that (p, r) is an edge, q is the one closest to p; or (ii) among all points $r \in \bar{\sigma}(q)$ such that (q, r) is an edge, p is the one closest to q. It is easy to see that every cone-separated pair (A_i, B_i) is still well connected in $\mathcal{S}^*$. Moreover, the graph $\mathcal{S}^*$ has $O(n/\varepsilon^{d-1})$ edges, since each edge can be charged to a unique combination of a point in P and a cone in $\mathcal{C}$.

Lemma 2.6 *In the graph $\mathcal{S}^* = (P, E_{\mathcal{S}^*})$, every cone-separated pair is well connected. Moreover, $\mathcal{S}^*$ has $O(n/\varepsilon^{d-1})$ edges, and every point has degree $O(\log^d n)$.*

2.2 Kinetic Maintenance of the Spanner

To summarize, our spanner works as follows. First we compute a CSPD Ψ_σ for every $\sigma \in \mathcal{C}$ using a range tree $\mathcal{T}_\sigma$. Then we produce a graph $\mathcal{S} = (P, E_\mathcal{S})$ such that every cone-separated pair (A_i, B_i) is well connected in $\mathcal{S}$. Finally, we obtain our linear-size spanner $\mathcal{S}^* = (P, E_{\mathcal{S}^*})$ by pruning some edges from $\mathcal{S}$; namely, for each cone σ and point p, we just add the edge $(p, q) \in E_\mathcal{S}$ to $E^*_\mathcal{S}$, where q is closest to p among the points inside $\sigma(p)$ that are connected to p in $\mathcal{S}$.

To kinetize our spanner, we therefore have to maintain the CSPD's Ψ_σ as the points move, and we have to maintain for each p and σ the closest point to p among those inside $\sigma(p)$ that are connected to p in $\mathcal{S}$.

Rank-Based Range Trees Fix a cone $\sigma \in \mathcal{C}$. Recall that the construction of Ψ_σ is based on a range tree $\mathcal{T}_\sigma$. Basch et al. [3] describe how to maintain a range tree in the kinetic setting: whenever two points exchange their order in one of the coordinates, delete and reinsert those points. Thus, all they need is a dynamic range tree. Unfortunately, the existing dynamic range trees—which use global or local rebuilding techniques [18], or the method of Willard and Lueker [23]—do not apply in our case, since they either only give amortized bounds or they require splitting and merging operations (which are hard for our CSPDs). Therefore, we take a different approach, which uses the rank-based technique that was also used to design kinetic BSPs [8] and kinetic kd-trees [1].

The basic idea is to define a skeleton tree which does not depend on the positions of the points. Since the structure of the skeleton does not depend on the positions of the points, it is static: no rebalancing operations are needed to maintain the skeleton as the points move. (One caveat: to save storage, certain parts of the skeleton are pruned, and which parts are pruned does depend on the positions.) In which canonical subsets of the skeleton tree a given point is stored, will depend only on the ranks of the coordinates of the point. This means the only events are when two points p and q swap

order along one of the coordinate axes. When that happens, p and q exchange their rank (in that coordinate)—the ranks of the other points are not influenced. Hence, to update the tree, we only need to delete and reinsert p and q with their new ranks. This changes only $O(\log^d n)$ canonical subsets. We then have to update the spanner edges defined for these canonical subsets. Below we make this idea precise.

Let $h_1, \ldots, h_d$ be the planes defining the cone σ. For each h_i, we define a coordinate axis x_i orthogonal to h_i and we let $x_i(p)$ denote the x_i-coordinate of a point p. To simplify the discussion, we assume that for two points $p, q \in P$, we have $x_i(p) \neq x_i(q)$ for all i. (Of course, coordinates will temporarily be equal when two points swap order, but the description below refers to the time intervals in between such events.) For a point $p \in P$, let $\text{rank}_i(p)$ denote the rank of $x_i(p)$ in the set $\{x_i(q) : q \in P\}$.

First we define the *skeleton* of our rank-based range tree, denoted $\mathcal{T}_\sigma^{\text{skel}}$. Let $n := |P|$. (In the kinetic setting, n is fixed.) The skeleton of a one-dimensional rank-based range tree is simply a balanced binary tree on n leaves, where the leaves correspond to ranks 1 to n from left to right; each internal node ν corresponds to a range of ranks, namely the ranks of all leaves in the subtree rooted at ν. The skeleton of a d-dimensional rank-based range tree consists of a main tree, which is also a balanced binary tree on n leaves, where each internal node ν has an associated structure $\mathcal{T}_\nu$ that is the skeleton of a $(d-1)$-dimensional rank-based range tree. Note that every tree in any level is a balanced binary tree on n ranks 1 to n independent of how many points in P lie in the tree. Therefore, the size of the skeleton of a d-dimensional rank-based range tree is $\Theta(n^d)$. Our rank-based range tree, however, uses only $O(n \log^d n)$ space. It will be obtained by pruning the skeleton, as described later. Note that we do not maintain or explicitly construct the skeleton; the skeleton is only used to define our rank-based range tree in an easy way.

Next, we describe where the points from P are stored in $\mathcal{T}_\sigma^{\text{skel}}$—this will give us the sets $P(\nu)$—and how the sets $R(\nu)$ are obtained, thus providing us with the pairs $(P(\nu), R(\nu))$ in the CSPD Ψ_σ.

We insert the points from P into $\mathcal{T}_\sigma^{\text{skel}}$ in the usual way. Thus, for a point $p \in P$, we follow the path in $\mathcal{T}_\sigma^{\text{skel}}$'s main tree to the leaf corresponding to $\text{rank}_i(p)$, and for each node ν on the search path, we recursively insert p into $\mathcal{T}_\nu$ (using the ranks $\text{rank}_2(p), \ldots, \text{rank}_d(p)$). For a node ν at any level in the tree, let $P(\nu)$ denote its canonical subset, which is the set of points p whose search path passes through ν. We explicitly store the sets $P(\nu)$ for the nodes in the tree at level d (that is, the trees defined for the dth coordinate). Observe that any point $p \in P$ is stored in $O(\log^d n)$ such d-level canonical subsets, which are distributed over $O(\log^{d-1} n)$ d-level trees.

Next, we search for each $p \in P$ with the range $\sigma(p)$ in $\mathcal{T}_\sigma^{\text{skel}}$, also in the usual way. Thus, we select in the main tree $O(\log n)$ nodes whose ranges together cover the x_1-range of $\sigma(p)$—more precisely, the rank_1-range—and we recursively search in the associated trees of these nodes. Notice that the canonical subsets $P(\nu)$ of the selected nodes at level-d trees together contain exactly the points inside $\sigma(p)$. For a node ν at any level, let $R(\nu)$ denote the set of ranges that select that node. The sets $R(\nu)$ for level-d nodes are explicitly stored with those nodes. As explained previously, the pairs $(P(\nu), R(\nu))$ for d-level nodes form the pairs in the CSPD Ψ_σ.

The total size of all sets $P(\nu)$ and $R(\nu)$ is $O(n \log^d n)$. However, the skeleton still has size $\Theta(n^d)$. Next, we show how to prune $\mathcal{T}_\sigma^{\text{skel}}$ to reduce its size to $O(n \log^d n)$.

Define a node v at any level in $\mathcal{T}_\sigma^{\text{skel}}$ to be *active* if one of the following holds:

(1) $P(v)$ is nonempty. (Note that $P(v)$ is necessarily nonempty if $P(v')$ is nonempty for some descendant v' of v.)
(2) $R(v)$ is nonempty, or there is a descendant v' of v such that $R(v')$ is nonempty.

A node that is not active is called *inactive*. We obtain our rank-based range tree $\mathcal{T}_\sigma^{\text{rb}}$ by pruning every inactive node of $\mathcal{T}_\sigma^{\text{skel}}(P)$. It is not hard to construct the rank-based range tree in $O(n \log^d n)$ time, by not first constructing the full skeleton, but only creating the active nodes as the points and ranges are inserted. This leads to the following lemma.

Lemma 2.7 *For any set P of n points in $\mathbb{R}^d$ and any canonical cone $\sigma \in \mathcal{C}$, the rank-based range tree $\mathcal{T}_\sigma^{\text{rb}}$ uses $O(n \log^d n)$ storage and can be constructed in $O(n \log^d n)$ time. Moreover, the size of the CSPD Ψ_σ based on $\mathcal{T}_\sigma^{\text{rb}}$ is $O(n \log^d n)$, and every point $p \in P$ appears in $O(\log^d n)$ pairs of Ψ_σ.*

Kinetic Maintenance We have described the rank-based range tree $\mathcal{T}_\sigma^{\text{rb}}$ and how to obtain the CSPD Ψ_σ from it. It remains to show how to maintain $\mathcal{T}_\sigma^{\text{rb}}$ and Ψ_σ (as well as the graph $\mathcal{S} = (P, E_\mathcal{S})$ constructed over Ψ_σ) as the points move.

The combinatorial structure of $\mathcal{T}_\sigma^{\text{rb}}$ depends only on the ranks of the points on the axes $x_1, \ldots, x_d$—it does not change as long as the order of the points along all axes remains the same. Hence we maintain, for each axis x_i, the points from P in an array $A_i[1..n]$ which is sorted on x_i-order. Thus $A_i[j]$ will contain the point p such that $\text{rank}_i(p) = j$. Now suppose that two points p, q swap their x_i-order. To handle such an event, we delete p and q, and then reinsert them with their new ranks. Note that a rank change for p also implies a change in the cone $\sigma(p)$. More precisely, the ranks of one of the planes bounding $\sigma(p)$ changes. The same is true for q. Hence, we also need to delete and reinsert $\sigma(p)$ and $\sigma(q)$.

These deletions and reinsertions of p, q, $\sigma(p)$, and $\sigma(q)$ do not change anything for the other points and cones, because their ranks are not influenced by the swap of p and q. Hence, the rank-based range tree remains unchanged except for $O(\log^d n)$ nodes that involve p and q. Insertions (and deletions) therefore involve two steps: determining those nodes and updating the set $E(P(v), R(v))$ of edges created for the cone-separated pairs for such nodes at level d.

Step 1: Finding the affected nodes.
This step is relatively straightforward. First, consider the insertion of a point p. We simply search in $\mathcal{T}_\sigma^{\text{rb}}$ and add p to the sets $P(v)$ for all nodes on the search path in d-level trees. The only slight complication is that sometimes the search path may proceed to a node that does not exist yet, because it was inactive. In such a case we simply create a new node with $\{p\}$ as its canonical subset. From this node we then proceed in the same way, creating the whole path to the leaf for p, and also creating associated structures—which will be single paths (with associated structures), etc.
The insertion of a range $\sigma(p)$ is done in a similar way: search with the range in $\mathcal{T}_\sigma^{\text{rb}}$ to determine the nodes v where $\sigma(p) \in R(v)$, creating new nodes where necessary. Deletions of points and ranges are done in a symmetric way (possibly deleting nodes that become inactive).

Step 2: Updating the sets $E(P(\nu), R(\nu))$.

Now consider a d-level node ν such that $P(\nu)$ or $R(\nu)$ change due to the insertion or deletion of a point or range. We describe how to insert a point into $P(\nu)$; insertions into $R(\nu)$ and deletions are handled in a similar fashion.

We only have to do something if both $P(\nu)$ and $R(\nu)$ are nonempty. Recall that to create the edge set $E(P(\nu), R(\nu))$ for the cone-separated pair $(P(\nu), R(\nu))$, the sets $P(\nu)$ and $R(\nu)$ are partitioned into a logarithmic number of groups $P(\nu)^j$ and $R(\nu)^j$. From a group $P(\nu)^j$ we then added edges to $R(\nu)^{j-1}$ in such a way that each $q \in R(\nu)^{j-1}$ receives only two edges (and vice versa).

To insert p into $P(\nu)$, we first determine the group $P(\nu)^j$ to which p belongs. To this end, we maintain $P(\nu)$ in sorted order in a tree—this can be done without any asymptotic overhead. If $P(\nu)^j$ is already full, then its last element, p', must move to the next group, $P(\nu)^{j+1}$. In this case, p can take over the edges from p', and we recursively insert p' into $P(\nu)^{j+1}$. The recursive insertion of p' into $P(\nu)^{j+1}$ may cause the last point, p'', from $P(\nu)^{j+1}$ to be inserted into $P(\nu)^{j+2}$, etc. The recursion ends when we insert a point into a nonfull group, (This could in fact be an empty group if the last group was already full. In this case we must create a new group.)

When we insert a point p into a nonfull group $P(\nu)^j$, there will be some point in $R(\nu)^{j-1}$ that does not have two "incoming" edges from $P(\nu)^j$ yet; we give p an edge to such a point, and we are done. (There is one special case here: when $R(\nu)^{j-1}$ is the last group among the $R(\nu)^i$'s, it could happen that the set $R(\nu)^{j-1}$ to which we want to connect is not full, or maybe does not even exist. Then we only need to take action if p is among the first $2|R(\nu)^{j-1}|$ points of $P(\nu)^j$.) To be able to quickly find such a point, we maintain a list of all points in $R(\nu)^i$ that do not yet have two incoming edges. (We do the same for $P(\nu)^i$ to deal with insertions into $R(\nu)$.) This extra information uses linear space in the size of $P(\nu)$ and is easy to maintain.

The whole procedure for inserting p into $P(\nu)$ works in $O(\log n)$ time.

Other Events There is one other type of event we must deal with, namely when two points in some set $P(\nu)$ or $R(\nu)$ exchange their order with respect to σ's representative edge. Note that this does imply a change to the tree $\mathcal{T}_\sigma^{\mathrm{rb}}$—indeed the representative edge of σ does not correspond to one of the axes for which $\mathcal{T}_\sigma^{\mathrm{rb}}$ is defined. To detect this type of event, we maintain for each σ an array $A_\sigma[1..n]$ on all points in P sorted according to their order with respect to σ's representative edge. When we have a swap between points p, q in this array, we check if there are any sets $P(\nu)$ or $R(\nu)$ that contain both p and q. All we then need to do is let them exchange their edges in $E(P(\nu), R(\nu))$.

Maintaining the Pruned Spanner $\mathcal{S}^* = (P, E_{\mathcal{S}^*})$ Above we described how to maintain the rank-based range tree $\mathcal{T}_\sigma^{\mathrm{rb}}$ and the graph $\mathcal{S} = (P, E_{\mathcal{S}})$ induced by it. However, to get a linear-size spanner, we need to maintain the pruned spanner $\mathcal{S}^* = (P, E_{\mathcal{S}^*})$. Recall that $\mathcal{S}^*$ is obtained by selecting, for each cone σ and point p, the edge $(p, q) \in \mathcal{S}$ such that q is closest to p among the points inside $\sigma(p)$ and adjacent to p in $\mathcal{S}$.

To maintain this edge kinetically, we observe that the selected edge for a point p and a cone σ can only change if two other points change order with respect to σ's representative edge—one of these points being the current selected point for p, the other being the new one. Thus, whenever two points q, r swap their order, we need to check for all points p such that (p, q) and (p, r) are edges in $E_{\mathcal{S}}$ whether we need to update p's selected edge. This can be done in $O(\log^d n)$ time, since the maximum degree in $\mathcal{S}$ is $O(\log^d n)$.

Putting it All Together In total, our kinetic spanner uses $d + 1$ sorted lists for each cone $\sigma \in \mathcal{C}$ to maintain the spanner $\mathcal{S}^*$ through time. Therefore, the number of events to be handled is $O(n^2/\varepsilon^{d-1})$ under the assumption that every point follows bounded-degree polynomials. As explained above, each event can be handled in $O(\log^{d+1} n)$ time. Moreover, each point is involved in at most two certificates per sorted list, so in $O(1/\varepsilon^{d-1})$ certificates in total. Hence, flight plan updates take only $O((\log n)/\varepsilon^{d-1})$ time. (The logarithmic factor is for updating the event queue when the failure times of the certificates has changed.)

Theorem 2.8 *For any $\varepsilon > 0$ and any set P of n points in $\mathbb{R}^d$, there is a kinetic $(1+\varepsilon)$-spanner of size $O(n/\varepsilon^{d-1})$ and maximum degree $O(\log^d n)$. Moreover, every point is involved in $O(1/\varepsilon^{d-1})$ certificates, and the number of events is $O(n^2/\varepsilon^{d-1})$ assuming the trajectories of the points can be described by bounded-degree polynomials. Each event can be handled in $O(\log^{d+1} n)$ time using a supporting data structure that needs $O((n/\varepsilon^{d-1})\log^d n)$ storage. A flight plan update takes $O((\log n)/\varepsilon^{d-1})$ time.*

Remark 2.9 Note that the bound on the number of events only uses the fact that any two points exchange their order along any one of the representative edges $O(1)$ times. This fact follows from the assumption that the trajectories are bounded-degree polynomials, but may hold in other settings as well.

3 Concluding Remarks

We have presented a new $(1 + \varepsilon)$-spanner for a set of points in $\mathbb{R}^d$. Our spanner has size $O(n/\varepsilon^{d-1})$ and can be maintained in $O(\log^{d+1} n)$ time per event as the points move. The number of events matches the $\Omega(n^2)$ lower bound from Gao et al. [11]. This is the first efficient kinetic $(1 + \varepsilon)$-spanner for which the number of events and the response time do not depend on the spread of the point set and that works for any fixed dimension. Unfortunately, the weight of our spanner can be much larger than the weight of a minimum spanning tree. We leave developing an efficient kinetic $(1+\varepsilon)$-spanner whose total weight is $O(wt(MST(P)))$ as a topic for further research.

References

1. Abam, M.A., de Berg, M., Speckmann, B.: Kinetic kd-trees and longest-side kd-trees. In: Proc. ACM Symposium on Computational Geometry, pp. 364–372 (2007)
2. Abam, M.A., de Berg, M., Gudmundsson, J.: A simple and efficient kinetic spanner. In: Proc. ACM Symposium on Computational Geometry, pp. 306–310 (2008)
3. Basch, J., Guibas, L.J., Zhang, L.: Proximity problems on moving points. In: Proc. ACM Symposium on Computational Geometry, pp. 344–351 (1997)
4. Basch, J., Guibas, L., Hershberger, J.: Data structures for mobile data. J. Algorithms **31**, 1–28 (1999)
5. Bose, P., Gudmundsson, J., Morin, P.: Ordered theta graphs. Comput. Geom. **28**, 11–18 (2004)
6. Chew, L.P.: There is a planar graph almost as good as the complete graph. In: Proc. ACM Symposium on Computational Geometry, pp. 169–177 (1986)
7. Clarkson, K.L.: Approximation algorithms for shortest path motion planning. In: Proc. ACM Symposium on Theory of Computing, pp. 56–65 (1987)
8. de Berg, M., Comba, J., Guibas, L.J.: A segment-tree based kinetic BSP. In: Proc. ACM Symposium on Computational Geometry, pp. 134–140 (2001)
9. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications, 3rd edn. Springer, Berlin (2008)
10. Eppstein, D.: Spanning trees and spanners. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 425–461. Elsevier, Amsterdam (2000)
11. Gao, J., Guibas, L.J., Nguyen, A.: Deformable spanners and applications. Comput. Geom. **35**(1–2), 2–19 (2006)
12. Gottlieb, L.-A., Roditty, L.: Improved algorithms for fully dynamic geometric spanners. In: Proc. ACM–SIAM Symposium on Discrete Algorithms, pp. 591–600 (2008)
13. Gottlieb, L.-A., Roditty, L.: An optimal dynamic spanner for doubling metric spaces. In: Proc. European Symposium on Algorithms, pp. 478–489 (2008)
14. Gudmundsson, J., Knauer, C.: Dilation and detour in geometric networks. In: Gonzalez, T. (ed.) Handbook on Approximation Algorithms and Metaheuristics. Chap. 52. Chapman & Hall/CRC, Boca Raton (2006)
15. Guibas, L.J.: Kinetic data structures: A state of the art report. In: Proc. Workshop on Algorithmic Foundations of Robotics, pp. 191–209 (1998)
16. Guibas, L.J.: Motion. In: Goodman, J., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, 2nd edn. pp. 1117–1134. CRC Press, Boca Raton (2004)
17. Keil, J.M.: Approximating the complete Euclidean graph. In: Proc. Scandinavian Workshop Algorithm Theory. Lecture Notes Computer Science, vol. 318, pp. 208–213. Springer, Berlin (1988)
18. Mehlhorn, K.: Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry. EATCS Monographs. Springer, Berlin (1984)
19. Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press, Cambridge (2007)
20. Roditty, L.: Fully dynamic geometric spanners. In: Proc. ACM Symposium on Computational Geometry, pp. 373–380 (2007)
21. Sharir, M., Agarwal, P.K.: Davenport–Schinzel Sequences and Their Geometric Applications. Cambridge University Press, Cambridge (1995)
22. Soares, J.: Graph spanners: A survey. Congr. Numer. **89**, 225–238 (1992)
23. Willard, D.E., Lueker, G.S.: Adding range restriction capability to dynamic data structures. J. ACM **32**(3), 597–617 (1985)