ON LOCALITY-SENSITIVE ORDERINGS AND THEIR APPLICATIONS*

TIMOTHY M. CHAN†, SARIEL HAR-PELED‡, AND MITCHELL JONES§

Abstract. For any constant d and parameter $\varepsilon \in (0, 1/2]$, we show the existence of (roughly) $1/\varepsilon^d$ orderings on the unit cube $[0,1)^d$, such that for any two points $p, q \in [0,1)^d$ close together under the Euclidean metric, there is a linear ordering in which all points between p and q in the ordering are "close" to p or q. More precisely, the only points that could lie between p and q in the ordering are points with Euclidean distance at most $\varepsilon \|p - q\|$ from either p or q. These orderings are extensions of the Z-order, and they can be efficiently computed.

Functionally, the orderings can be thought of as a replacement to quadtrees and related structures (like well-separated pair decompositions). We use such orderings to obtain surprisingly simple algorithms for a number of basic problems in low-dimensional computational geometry, including (i) dynamic approximate bichromatic closest pair, (ii) dynamic spanners, (iii) dynamic approximate minimum spanning trees, (iv) static and dynamic fault-tolerant spanners, and (v) approximate nearest neighbor search.

Key words. Approximation algorithms, data structures, computational geometry.

AMS subject classifications. 68W25, 68P05

1. Preface.

In this paper, we describe a technique that leads to new, simpler algorithms for a number of fundamental proximity problems in low-dimensional Euclidean spaces.

Given data, having an ordering over it is quite useful—it enables one to sort it, store it, and search it efficiently, among other things. Such an order is less natural for points in the plane (or in higher dimensions). One way to impose such orders is by using bijective mappings from the plane to the line (which has a natural order, and thus endows the plane with an order). Such mappings, known as space-filling curves, were discovered in 1890 by Peano [32]. (See also the book by Sagan [34] for more information on space-filling curves.) For computational purposes, the Z-order, a somewhat inferior space-filling curve, is the easiest to implement as it is easily computed by interleaving the bits of the x and y coordinates.

A natural property one desires in an ordering of the plane is that it preserves locality—points that are close together geometrically remain close in the resulting ordering. Unfortunately, no mapping/ordering can have this property universally, as the topology of the line and the plane are fundamentally different. Nevertheless, Z-order already has some nice locality properties—it maps certain squares to intervals on the real line, and these squares forms grids that cover the unit square. Furthermore, these grids are universal, in the sense that there is a grid for any desired resolution.

To get better locality properties, one has to use more orders. It is known that

*Submitted to the editors 02/22/19. A preliminary version of this paper appeared in ITCS 2019 [11].

†Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 (tmc@illinois.edu).

Funding: Work on this paper was partially supported by NSF AF award CCF-1814026.

‡Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 (sariel@illinois.edu).

Funding: Work on this paper was partially supported by NSF AF awards CCF-1421231 and CCF-.

§Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801 (mfjones2@illinois.edu).

if one uses three orders in the plane (which is the result of shifting the plane before applying the Z-order), then for any axis parallel square $\mathcal{C}$ (inside the unit square), there exists a square $\mathcal{C}'$ that contains $\mathcal{C}$, such that $\mathcal{C}'$ is only slightly bigger than $\mathcal{C}$, and one of the three orders maps $\mathcal{C}'$ to an interval.

Out purpose here is to get an even stronger locality property, which requires a larger collection of orderings. Specifically, consider two points $p, p' \in [0,1]^2$. The desired property is that there are two squares $\mathcal{C}$ and $\mathcal{C}'$, and an order σ in the collection, with the following properties: (i) $p \in \mathcal{C}$ and $p' \in \mathcal{C}'$, (ii) the diameters of $\mathcal{C}$ and $\mathcal{C}'$ are only an ε-fraction of the distance between p and p', (iii) $\mathcal{C}$ and $\mathcal{C}'$ are mapped to two intervals on the real line by σ, and (iv) these two intervals are adjacent. Such an ordering σ with the desired properties is illustrated in Figure 1.1.

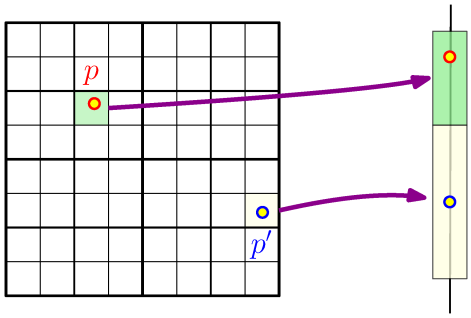

Figure 1.1

For algorithmic applications, this collection of orders need to be small, and it needs to be easily computable. Surprisingly, we show that the desired collection of orders has size that depends only on ε, and these orders can be easily computed.

To see why having such a collection of orders is so useful, consider the problem of computing the closest pair of points in a given set of points P. Every order in the collection induces an ordering of P. Furthermore, the closest pair of points are going to be adjacent in one of these orders, and as such can be readily computed by considering all consecutive pairs of points in the ordering (the number of such pairs is linear). Furthermore, using balanced binary search trees, it is easy to maintain each ordered set under insertions and deletions. Therefore, one can maintain the closest pair of points by storing P in such a data structure for each of the orderings. As a result, a dynamic problem that might seem in advance somewhat challenging reduces (essentially) to the mundane task of maintaining ordered sets under insertions and deletions.

2. Introduction.

Quadtrees and Z*-order.* Consider a point set $P \subseteq [0,1)^2$, its quadtree, and a depth-first search (DFS) traversal of this quadtree. One can order the points of P according to this traversal, resulting in some ordering $\prec$ of the underlying set $[0,1)^2$. The relation $\prec$ is the ordering along some space filling mapping.

One particular ordering of interest is the ***Z-order***. Conceptually speaking, the Z-order can be thought of as a DFS of the quadtree over $[0,1)^2$, where the children of each node in the quadtree are always visited in the same pre-defined order (see Figure 1.2). The Z-order is a total ordering over the points in $[0,1)^2$, and can be formally defined by a bijection z from the unit interval $[0,1)$ to the unit square $[0,1)^2$. Given a real

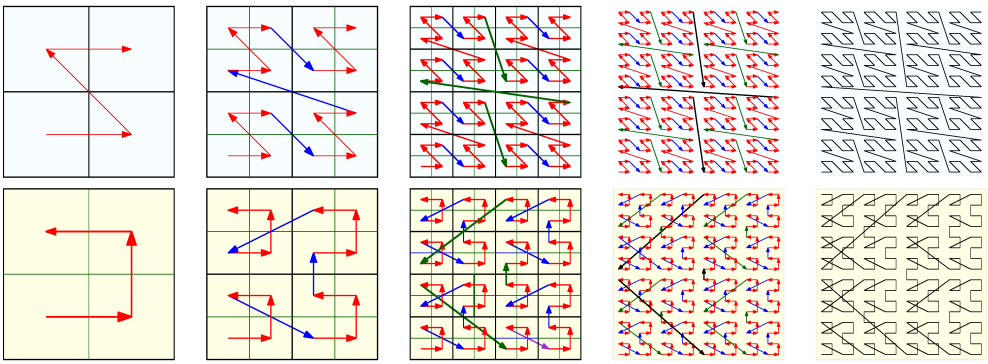

Figure 1.2: Changing the order in which a DFS visits the children of a quadtree node induces a different ordering of the underlying square (and produces different space filling curves). The top row shows the Z-order (or $\aleph$-order), and the bottom row shows the $\supset$-order.

number $\alpha \in [0, 1)$, with the binary expansion $\alpha = 0.x_1x_2x_3\ldots$ (i.e., $\alpha = \sum_{i=1}^{\infty} x_i 2^{-i}$), the Z-order mapping of α is the point $z(\alpha) = (0.x_2x_4x_6\ldots, 0.x_1x_3x_5\ldots)$. We note that the Z-order mapping z is not continuous. Nevertheless, the Z-order mapping has the advantage of being easy to define. In particular, computing the Z-order or its inverse is quite easy, if one is allowed bitwise-logical operations—in particular, the ability to compute compressed quadtrees efficiently is possible only if such operations are available [20]. The approach extends to higher constant dimensions.

The idea of using the Z-order can be traced back to the work of Morton [31], and it is widely used in databases and seems to improve performance in practice [25]. Once comparison by Z-order is available, building a compressed quadtree is no more than storing the points according to the Z-order, and this yields simple data structures for various problems. For example, Liao et al. [28] and Chan [8, 9, 10] applied the Z-order to obtain simple efficient algorithms for approximate nearest neighbor search and related problems.

Shifting. The Z-order (and quadtrees) does not preserve distance. That is, two points that are far away might be mapped to two close-together points, and vice versa. This problem is even apparent when using a grid, where points that are close together get separated into different grid cells. One way to get around this problem is to shift the grid (deterministically or randomly) [22]. The same approach works for quadtrees—one can shift the quadtree constructed for a point set several times such that for any pair of points in the quadtree, there will be a shift where the two points are in a cell of diameter that is $O_d(1)$ times their distance. (Throughout, we use the O_d notation to hide constants that depend on d. Similarly, O_ε hides dependencies on ε.) Improving an earlier work by Bern [3], Chan [7] showed that $2\lceil d/2\rceil + 1$ deterministic shifts are enough in d dimensions (a proof is reproduced in Appendix A.2). A somewhat similar shifting scheme was also suggested by Feige and Krauthgamer [16]. Random shifting of quadtrees underlines, for example, the approximation algorithm by Arora for Euclidean TSP [2].

By combining Z-order with shifting, both Chan [8] and Liao et al. [28] observed an extremely simple data structure for $O_d(1)$-approximate nearest neighbor search in constant dimensions: just store the points in Z-order for each of the $2\lceil d/2\rceil + 1$ shifts;

given a query point q, find the successor and predecessor of q in the Z-order by binary search for each of the shifts, and return the closest point found. The data structure can be easily made dynamic to support insertions and deletions of points, and can also be adapted to find $O_d(1)$-approximate bichromatic closest pairs.

For approximate nearest neighbor (ANN) search, the $O_d(1)$ approximation factor can be reduced to $1+\varepsilon$ for any fixed $\varepsilon > 0$, though the query algorithm becomes more involved [8] and unfortunately cannot be adapted to compute $(1+\varepsilon)$-approximate bichromatic closest pairs dynamically. (In the monochromatic case, however, the approach can be adapted to find *exact* closest pairs, by considering $O_d(1)$ successors and predecessors of each point [8].)

For other proximity-related problems such as spanners and approximate minimum spanning trees (MST), this approach does not seem to work as well: for example, the static algorithms in [10], which use the Z-order, still requires explicit constructions of compressed quadtrees and are not easily dynamizable.

Main new technique: Locality-sensitive orderings. For any given $\varepsilon > 0$, we show that there is a family of $O_d((1/\varepsilon^d)\log(1/\varepsilon))$ orderings of $[0,1)^d$ with the following property: For any $p, q \in [0,1)^d$, there is an ordering in the family such that all points lying between p and q in this ordering are within distance at most $\varepsilon\,\|p-q\|$ from either p or q (where $\|\cdot\|$ is the standard Euclidean norm). The order between two points can be determined efficiently using some bitwise-logical operations. See Theorem 3.10. We refer to these as ***locality-sensitive orderings***. They generalize the previous construction of $2\lceil d/2\rceil + 1$ shifted copies of the Z-order, which guarantees the stated property only for a large specific constant (equivalent to setting $\varepsilon \approx d^{3/2}$). The new refined property ensures, for example, that a $(1+\varepsilon)$-approximate nearest neighbor of a point q can be found among the immediate predecessors and successors of q in these orderings.

Applications. Locality-sensitive orderings immediately lead to simple algorithms for a number of problems, as listed below. Many of these results are significant simplification of previous work; some of the results are new.

(A) Approximate bichromatic closest pair. Theorem 4.2 presents a data structure that maintains a $(1+\varepsilon)$-approximate closest bichromatic pair for two sets of points in $\mathbb{R}^d$, with an update time of $O_{d,\varepsilon}(\log n)$, for any fixed $\varepsilon > 0$ (the hidden factors depending on ε are proportional to $(1/\varepsilon^d)\log^2(1/\varepsilon)$). Previously, a general technique of Eppstein [14] can be applied in conjunction with a dynamic data structure for ANN, but the amortized update time increases by two $\log n$ factors.

(B) Dynamic spanners. For a parameter $t \geq 1$ and a set of points P in $\mathbb{R}^d$, a graph $G = (P, E)$ is a t-spanner for P if for all $p, q \in P$, there is a p-q path in G of length at most $t\,\|p-q\|$. Static algorithms for spanners have been extensively studied in computational geometry. The dynamic problem appears tougher, and has also received much attention (see Table 2.1). We obtain a very simple data structure for maintaining dynamic $(1+\varepsilon)$-spanners in Euclidean space with an update (insertion and deletion) time of $O_{d,\varepsilon}(\log n)$ and having $O_{d,\varepsilon}(n)$ edges in total, for any fixed $\varepsilon > 0$. See Theorem 4.4. Although Gottlieb and Roditty [19] have previously obtained the same update time $O_{d,\varepsilon}(\log n)$, their method requires much more intricate details. (Note that Gottlieb and Roditty's method more generally applies to spaces with bounded doubling dimension, but no simpler methods have been reported in the Euclidean setting.)

(C) Dynamic approximate minimum spanning trees. As is well-known [5, 20], a $(1+\varepsilon)$-

reference	insertion time	deletion time
Roditty [33]	$\log n$	$n^{1/3}\log^{O(1)} n$
Gottlieb and Roditty [18]	$\log^2 n$	$\log^3 n$
Gottlieb and Roditty [19]	$\log n$	$\log n$
Theorem 4.4	$\log n$	$\log n$

Table 2.1: Previous work and our result on dynamic $(1+\varepsilon)$-spanners in $\mathbb{R}^d$. All bounds are of the form $O_{d,\varepsilon}(\cdot)$ (the hidden dependencies on ε are $1/\varepsilon^{O(d)}$).

approximate Euclidean MST of a point set P can be computed from the MST of a $(1+\varepsilon)$-spanner of P. In our dynamic spanner (and also Gottlieb and Roditty's method [19]), each insertion/deletion of a point causes $O_{d,\varepsilon}(1)$ edge updates to the graph. Immediately, we thus obtain a dynamic data structure for maintaining a $(1+\varepsilon)$-approximate Euclidean MST, with update time (ignoring dependencies on d and ε) equal to that for the dynamic graph MST problem, which is currently $O(\log^4 n/\log\log n)$ with amortization [23].

(D) Static and dynamic vertex-fault-tolerant spanners. For parameters $k, t \geq 1$ and a set of points P in $\mathbb{R}^d$, a k-vertex-fault-tolerant t-spanner is a graph G which is a t-spanner and for any $P' \subseteq P$ of size at most k, the graph $G \setminus P'$ remains a t-spanner for $P \setminus P'$. Fault-tolerant spanners have been extensively studied (see Table 2.2). Locality-sensitive orderings lead to a very simple construction for k-vertex-fault-tolerant $(1+\varepsilon)$-spanners, with $O_{d,\varepsilon}(kn)$ edges, maximum degree $O_{d,\varepsilon}(k)$, and $O_{d,\varepsilon}(n\log n + kn)$ running time. See Theorem 4.6. Although this result was known before, all previous constructions (including suboptimal ones), from Levcopoulos et al.'s [27] to Solomon's work [35], as listed in Table 2.2, require intricate details. It is remarkable how effortlessly we achieve optimal $O_{d,\varepsilon}(k)$ degree, compared to the previous methods. (Note, however, that some of the more recent previous constructions more generally apply to spaces with bounded doubling dimension, and some also achieve good bounds on other parameters such as the total weight and the hop-diameter.)

Our algorithm can be easily made dynamic, with $O_{d,\varepsilon}(\log n + k)$ update time. No previous results on dynamic fault-tolerant spanners were known.

(E) Approximate nearest neighbors. Locality-sensitive orderings lead to a simple dynamic data structure for $(1+\varepsilon)$-approximate nearest neighbor search with $O_{d,\varepsilon}(\log n)$ time per update/query. While this result is not new [8], we emphasize that the query algorithm is the simplest so far—it is just a binary search in the orderings maintained.

Computational models and assumptions. The model of computation we have assumed is a unit-cost real RAM, supporting standard arithmetic operations and comparisons (but no floor function), augmented with bitwise-logical operations (bitwise-exclusive-or and bitwise-and), which are commonly available in programming languages (and in reality are cheaper than some arithmetic operations like multiplication).

If we assume that input coordinates are integers bounded by U and instead work in the word RAM model with $(\log U)$-bit words ($U \geq n$), then our approach can actually yield *sublogarithmic* query/update time. For example, we can achieve $O_{d,\varepsilon}(\log\log U)$

reference	# edges	degree	running time
Levcopoulos et al. [27]	$2^{O(k)}n$	$2^{O(k)}$	$n\log n + 2^{O(k)}n$
	k^2n	unbounded	$n\log n + k^2n$
	$kn\log n$	unbounded	$kn\log n$
Lukovszki [29, 30]	kn	k^2	$n\log^{d-1} n + kn\log\log n$
Czumaj and Zhao [13]	kn	k	$kn\log^d n + k^2n\log k$
H. Chan et al. [6]	k^2n	k^2	$n\log n + k^2n$
Kapoor and Li [26]/Solomon [35]	kn	k	$n\log n + kn$
Theorem 4.6	kn	k	$n\log n + kn$

Table 2.2: Previous work and our result on static k-vertex-fault-tolerant $(1+\varepsilon)$-spanners in $\mathbb{R}^d$. All bounds are of the form $O_{d,\varepsilon}(\cdot)$ (the hidden dependencies on ε are $1/\varepsilon^{O(d)}$).

expected time for dynamic approximate bichromatic closest pair, dynamic spanners, and dynamic ANN, by replacing binary search with van Emde Boas trees [36]. Sublogarithmic algorithms were known before for dynamic ANN [8], but ours is the first sublogarithmic result for dynamic $(1+\varepsilon)$-spanners. Our results also answers the open problem of dynamic $(1+\varepsilon)$-approximate bichromatic closest pair in sublogarithmic time, originally posed by Chan and Skrepetos [12].

Throughout, we assume (without loss of generality) that ε is a power of 2; that is, $\varepsilon = 2^{-E}$ for some positive integer E.

3. Locality-sensitive orderings.

3.1. Grids and orderings.

DEFINITION. *For a set X, consider a total order (or ordering) $\prec$ on the elements of X. Two elements $x, y \in X$ are* **adjacent** *if there is no element $z \in X$, such that $x \prec z \prec y$ or $y \prec z \prec x$.*

Given two elements $x, y \in X$, such that $x \prec y$, the **interval** *$[x, y)$ is the set $[x, y) = \{x\} \cup \{z \in X \mid x \prec z \prec y\}$.*

The following is well known, and goes back to a work by Walecki in the 19th century [1]. We include a proof in Appendix A.1 for the sake of completeness. (If we don't care about the constant factor in the number of orderings, there are other straightforward alternative proofs.)

LEMMA 3.1. *For n elements $\{0, \ldots, n-1\}$, there is a set $\mathfrak{O}$ of $\lceil n/2 \rceil$ orderings of the elements, such that, for all $i, j \in \{0, \ldots, n-1\}$, there exists an ordering $\sigma \in \mathfrak{O}$ in which i and j are adjacent.*

DEFINITION 3.2. *Consider an axis-parallel cube $\mathcal{C} \subseteq \mathbb{R}^d$ with side length ℓ. Partitioning it uniformly into a $t \times t \times \cdots \times t$ grid G creates the* **t-grid** *of $\mathcal{C}$. The grid G is a set of t^d identically sized cubes with side length ℓ/t.*

For a cube $\square \subseteq \mathbb{R}^d$, its **diameter** *is $\mathsf{diam}(\square) = \mathsf{sidelength}(\square)\sqrt{d}$.*

By Lemma 3.1 we obtain the following result.

COROLLARY 3.3. *For a t-grid G of an axis-parallel cube $\mathcal{C} \subseteq \mathbb{R}^d$, there is a set $\mathfrak{O}(t, d)$ of $O(t^d)$ orderings, such that for any $\square_1, \square_2 \in \mathsf{G}$, there exists an order $\sigma \in \mathfrak{O}(t, d)$ where $\square_1$ and $\square_2$ are adjacent in σ.*

3.2. ε-Quadtrees.

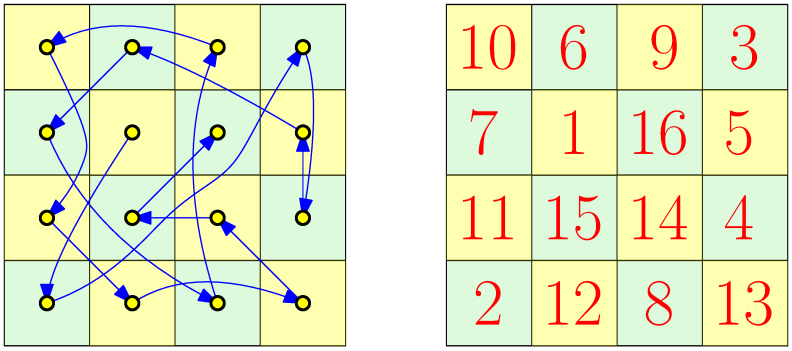

Figure 3.1: One ordering of a set of cells.

DEFINITION 3.4. *An* **ε-quadtree** *$\mathcal{T}_\varepsilon$ is a quadtree-like structure, built on a cube with side length ℓ, where each cell is partitioned into a $(1/\varepsilon)$-grid. The construction then continues recursively into each grid cell of interest. As such, a node in this tree has up to $1/\varepsilon^d$ children, and a node at level $i \geq 0$ has an associated cube of side length $\ell\varepsilon^i$. When $\varepsilon = 1/2$, this is a regular quadtree.*

LEMMA 3.5. *Let $E > 0$ be an integer, $\varepsilon = 2^{-E}$, and $\mathcal{T}$ be a regular quadtree over $[0,2)^d$. There are ε-quadtrees $\mathcal{T}_\varepsilon^0, \ldots, \mathcal{T}_\varepsilon^{E-1}$, such that the collection of cells at each level in $\mathcal{T}$ is contained in exactly one of these ε-quadtrees.*

Proof. For $i = 0, \ldots, E-1$, construct the ε-quadtree $\mathcal{T}_\varepsilon^i$ using the cube $\left[0, 2^{E-i+1}\right)^d \supseteq [0,2)^d$ as the root. Now for $j \in \{0, \ldots, E-1\}$, observe that the collection of cells at levels $j, j+E, j+2E, \ldots,$ of $\mathcal{T}$ will also be in the quadtree $\mathcal{T}_\varepsilon^j$. Indeed, any node at level $j + \ell E$ in $\mathcal{T}$ corresponds to a cell of side length $2^{-(j+\ell E)+1}$. Now in the $(\ell+1)$th level of quadtree $\mathcal{T}_\varepsilon^j$, this same node will have side length $\varepsilon^{\ell+1} 2^{E-j+1} = 2^{-(j+\ell E)+1}$. ∎

Consider an ε-quadtree $\mathcal{T}_\varepsilon$. Every node has up to $1/\varepsilon^d$ children. Consider any ordering σ of $\left\{1, \ldots, 1/\varepsilon^d\right\}$. Conceptually speaking, this induces a DFS of $\mathcal{T}_\varepsilon$ that always visits the children of a node in the order specified by σ. This induces an ordering on the points in the cube which is the root of $\mathcal{T}_\varepsilon$. Indeed, for any two points, imagine storing them in an ε-quadtree—this implies that the two points are each stored in their own leaf node, which contains no other point of interest. Now, independently of what other points are stored in the quadtree, this DFS traversal would visit these two points in the same order. This can be viewed as a space filling curve (which is not continuous) which maps a cube to an interval. This is a generalization of the Z-order. In particular, given a point set stored in $\mathcal{T}_\varepsilon$, and given σ, one can conceptually order the points according to this DFS traversal, resulting in 1-dimensional ordering of the points. We denote the resulting ordering by $(\mathcal{T}_\varepsilon, \sigma)$.

In Section 3.3, we show that given $(\mathcal{T}_\varepsilon, \sigma)$, the order of any two points in $[0,2)^d$ can be determined efficiently, and avoids explicitly handling this DFS traversal of $\mathcal{T}_\varepsilon$. Alternatively, the DFS on $\mathcal{T}_\varepsilon$ (according to σ) is implicitly defined by the total ordering $(\mathcal{T}_\varepsilon, \sigma)$ of points in $[0,2)^d$.

DEFINITION 3.6. *Let Π be the set of all orderings of $[0,2)^d$, induced by picking one of the $\lg(1/\varepsilon)$ trees of Lemma 3.5, together with an ordering $\sigma \in \mathfrak{O}(1/\varepsilon, d)$, as defined by Lemma 3.1. Each ordering in Π is an* **ε-ordering**.

Suppose there are two points which lie in a quadtree cell that has diameter close to their distance. Formally, consider two points $p, q \in [0,1)^d$, a parameter $\varepsilon > 0$, such that p, q are both contained in a cell $\square$ of the regular quadtree $\mathcal{T}$ with $\mathsf{diam}(\square) \leq 2\|p-q\|$. Then, there is an ε-quadtree $\mathcal{T}_\varepsilon$ that has $\square$ as a node, and let $\square_p$ and $\square_q$ be the two children of $\square$ in $\mathcal{T}_\varepsilon$, containing p and q respectively. Furthermore, there is an ordering $\sigma \in \mathfrak{O}(1/\varepsilon, d)$, such that $\square_p$ and $\square_q$ are adjacent. As such, the cube $\square_p$ (resp., $\square_q$) corresponds to an interval $[x, x')$ (resp., $[x', x'')$) in the ordering $(\mathcal{T}_\varepsilon, \sigma)$, and these two intervals are adjacent. In particular, this implies that all points lying between p and q in σ have distance at most $2\varepsilon\|p-q\|$ from p or q.

If the above statement were true for all pairs of points, then this would imply the main result (Theorem 3.10). However, consider the case when there are two points close together, but no appropriately sized quadtree cell contains both p and q. In other words, two points that are close together might get separated by nodes that are much bigger in the quadtree, and this would not provide the guarantee of the main result. However, this issue can be resolved using shifting. We need the following result of Chan [7, Lemma 3.3]—a proof is provided in Appendix A.2.

LEMMA 3.7. *Consider any two points $p, q \in [0,1)^d$, and let $\mathcal{T}$ be the infinite quadtree of $[0,2)^d$. For $D = 2\lceil d/2 \rceil$ and $i = 0, \ldots, D$, let $v_i = (i/(D+1), \ldots, i/(D+1))$. Then there exists an $i \in \{0, \ldots, D\}$, such that $p + v_i$ and $q + v_i$ are contained in a cell of $\mathcal{T}$ with side length $\leq 2(D+1)\|p-q\|$.*

3.3. Comparing two points according to an ε-ordering. We now show how to efficiently compare two points in P according to a given ε-ordering σ with a shift v_i. The shift can be added to the two points directly, and as such we can focus on comparing two points according to σ.

First, we show how to compare the msb of two numbers using only bitwise-exclusive-or and bitwise-and operations. We remark that Observation 3.8 (A) is from Chan [8].

OBSERVATION 3.8. *Let $\oplus$ denote the bitwise-exclusive-or operator. Define $\mathrm{msb}(a) := -\lfloor \lg a \rfloor$ to be the index of the most significant bit in the binary expansion of $a \in [0,2)$. Given $a, b \in [0,2)$, one can compare the* msb *of two numbers using the following:*

(A) $\mathrm{msb}(a) > \mathrm{msb}(b)$ *if and only if $a < b$ and $a < a \oplus b$.*

(B) $\mathrm{msb}(a) = \mathrm{msb}(b)$ *if and only if $a \oplus b \leq a \wedge b$, where $\wedge$ is the bitwise-and operator.*

Proof. (A) Observe that if $\mathrm{msb}(a) > \mathrm{msb}(b)$, then $2^{-\mathrm{msb}(a)} \leq a < 2^{-\mathrm{msb}(a)+1} \leq 2^{-\mathrm{msb}(b)} \leq b$. Since $\mathrm{msb}(a) > \mathrm{msb}(b)$ and $a < b$, we have $\mathrm{msb}(a \oplus b) = \mathrm{msb}(b)$. As such, we have $a < 2^{-\mathrm{msb}(a)+1} \leq 2^{-\mathrm{msb}(b)} = 2^{-\mathrm{msb}(a \oplus b)} \leq a \oplus b$.

Assume that $a < b$ and $a < a \oplus b$. Since $a < b$, it must be that $\mathrm{msb}(a) \geq \mathrm{msb}(b)$. Observe that if $\mathrm{msb}(a) = \mathrm{msb}(b)$, then $a \oplus b < a$, which is impossible. It follows that $\mathrm{msb}(a) > \mathrm{msb}(b)$, as desired.

(B) Follows by applying (A) twice (in addition to using the inequalities $a \wedge b \leq a$ and $a \wedge b \leq b$), one can show that $a \oplus b \leq a \wedge b$ if and only if $\mathrm{msb}(a) \geq \mathrm{msb}(b)$ and $\mathrm{msb}(b) \geq \mathrm{msb}(a)$. ∎

LEMMA 3.9. *Let $p = (p_1, \ldots, p_d)$ and $q = (q_1, \ldots, q_d)$ be two distinct points in $P \subseteq [0,2)^d$ and $\sigma \in \Pi$ be an ε-ordering over the cells of some ε-quadtree $\mathcal{T}_\varepsilon$ storing P. Then one can determine if $p \prec_\sigma q$ using $O(d \log(1/\varepsilon))$ bitwise-logical operations.*

Proof. Recall ε is a power of two and $E = \lg(1/\varepsilon)$. In order to compare p and q, for

$i = 1, \ldots, d$, compute $a_i = p_i \oplus q_i$. Find an index i' such that $\mathrm{msb}(a_{i'}) \leq \mathrm{msb}(a_i)$ for all i. Such an index can be computed with $O(d)$ msb comparisons (using Observation 3.8 (A)). Given $p_{i'}$ and $q_{i'}$, we next determine the place in which $p_{i'}$ and $q_{i'}$ first differ in their binary representation. Note that because ε is a power of two, each digit in the base $1/\varepsilon$ expansion of $p_{i'}$ corresponds to a block of E bits in the binary expansion of $p_{i'}$. Suppose that $p_{i'}$ and $q_{i'}$ first differ inside the hth block at an index $j \in \{1, \ldots E\}$.

The algorithm now locates this index j. To do so, for $j = 1, \ldots, E$, let $b_j = 2^{E-j}/(2^E - 1) \in (0, 1]$ be the number whose binary expansion has a 1 in positions $j, j + E, j + 2E, \ldots$, and 0 everywhere else. For $j = 1, \ldots, E$, compute $b_j \wedge a_{i'}$ and check if $\mathrm{msb}(a_{i'}) = \mathrm{msb}(b_j \wedge a_{i'})$ (using Observation 3.8 (B)). When the algorithm finds such an index j obeying this equality, it exits the loop. We know that $p_{i'}$ and $q_{i'}$ first differ in the jth position inside the hth block (the value of h is never explicitly computed).

It remains to extract the E bits from the hth block in each coordinate $p_1, \ldots, p_d$. For $i = 1, \ldots, d$, let $B_i \in \{0, 1\}^E$ be the bits inside the hth block of p_i. For $k = 1, \ldots, E$, set $B_{i,k} = \mathbb{1}\left[\mathrm{msb}(2^{j-k} a_{i'}) = \mathrm{msb}((2^{j-k} a_{i'}) \wedge p_i)\right]$ (where $\mathbb{1}[\cdot]$ is the indicator function). By repeating a similar process for all $q_1, \ldots, q_d$, we obtain the coordinates of the cells in which p and q differ. We can then consult σ to determine whether or not $p \prec_\sigma q$.

This implies that p and q can be compared using $O(d \log(1/\varepsilon))$ operations by Observation 3.8. ∎

Remark. In the word RAM model for integer input, the extra $\log(1/\varepsilon)$ factor in the above time bound can be eliminated: msb can be explicitly computed in $O(1)$ time by a complicated algorithm of Fredman and Willard [17]; this allows us to directly jump to the right block of each coordinate and extract the relevant bits. (Furthermore, assembly operations performing such computations are nowadays available on most CPUs.)

3.4. The result.

THEOREM 3.10. *For $\varepsilon \in (0, 1/2]$, there is a set Π^+ of $O_d(\log(1/\varepsilon)/\varepsilon^d)$ orderings of $[0, 1)^d$, such that for any two points $p, q \in [0, 1)^d$ there is an ordering $\sigma \in \Pi^+$ defined over $[0, 1)^d$, such that for any point u with $p \prec_\sigma u \prec_\sigma q$ it holds that either $\|p - u\| \leq \varepsilon \|p - q\|$ or $\|q - u\| \leq \varepsilon \|p - q\|$.*

Furthermore, given such an ordering σ, and two points p, q, one can compute their ordering, according to σ, using $O(d \log(1/\varepsilon))$ arithmetic and bitwise-logical operations.

Proof. Let Π^+ be the set of all orderings defined by picking an ordering from Π, as defined by Definition 3.6 using the parameter ε, together with a shift from Lemma 3.7.

Consider any two points $p, q \in [0, 1)^d$. By Lemma 3.7 there is a shift for which the two points fall into a quadtree cell $\square$ with side length at most $2(D+1)\|p - q\|$. Next, there is an ε-quadtree $\mathcal{T}_\varepsilon$ that contains $\square$, and the two children that correspond to two cells $\square_p$ and $\square_q$ with side length at most $2(D+1)\varepsilon\|p - q\|$, which readily implies that the diameter of these cells is at most $2(D+1)\sqrt{d}\varepsilon\|p - q\|$. Furthermore, there is an ε-ordering in Π such that all the points of $\square_p$ are adjacent to all the points of $\square_q$ in this ordering. This implies the desired claim, after adjusting ε by a factor of $2(D+1)\sqrt{d}$ (and rounding to a power of 2). ∎

From now on, we refer to the set of orderings Π^+ in the above Theorem as locality-sensitive orderings. We remark that by the readjustment of ε in the final step of the proof, the number of locality-sensitive orderings when including the factors involving d is $O(d^{3/2})^d \cdot (1/\varepsilon^d) \log(1/\varepsilon)$.

347 **3.4.1. Discussion.**

348 *Connection to locality-sensitive hashing.* Let P be a set of n points in Hamming
349 space $\{0,1\}^d$. Consider the decision version of the $(1+\varepsilon)$-approximate nearest neigh-
350 bor problem. Specifically, for a pre-specified radius r and any given query point q,
351 we would like to efficiently decide whether or not there exists a point $p \in P$ such
352 that $\|q-p\|_1 \leq (1+\varepsilon)r$ or conclude that all points in P are at least distance r from
353 q. The locality-sensitive hashing (LSH) technique [24] implies the existence of a data
354 structure supporting this type of decision query in time $O(dn^{1/(1+\varepsilon)} \log n)$ time (which
355 is correct with high probability) and using total space $O(dn^{1+1/(1+\varepsilon)} \log n)$. Similar
356 results also hold in the Euclidean setting.

357 At a high level, LSH works as follows. Start by choosing $k := k(\varepsilon, r, n)$ indices in
358 $[d]$ at random (with replacement). Let R denote the resulting multiset of coordinates.
359 For each point $p \in P$, let p_R be the projection p onto these coordinates of R. We can
360 group the points of P into buckets, where each bucket contains points with the same
361 projection. Given a query point q, we check if any of the points in the same bucket
362 as q is at distance at most $(1+\varepsilon)r$ from q. This construction can also be repeated a
363 sufficient number of times in order to guarantee success with high probability.

364 The idea of bucketing can also be viewed as an implicit ordering on the randomly
365 projected point set by ordering points lexicographically according to the k coordinates.
366 In this sense, the query algorithm can be viewed as locating q within each of the
367 orderings, and comparing q to similar points nearby in each ordering. From this
368 perspective, every locality-sensitive ordering can be viewed as an LSH scheme. Indeed,
369 for a given query point q, the approximate nearest neighbor to q can be found by
370 inspecting the elements adjacent to q in each of the locality-sensitive orderings and
371 returning the closest point to q found (see Theorem 4.7).

372 Of course, the main difference between the two schemes is that for every fixed ε,
373 the number of "orderings" in an LSH scheme is polynomial in both d and n. While for
374 locality-sensitive orderings, the number of orderings remains exponential in d. This
375 trade-off is to be expected, as locality-sensitive orderings guarantee a much stronger
376 property than that of an LSH scheme.

377 *Extension of locality-sensitive orderings to other norms in Euclidean space.* The
378 L_p-norm, for $p \geq 1$, of a vector $x \in \mathbb{R}^d$ is defined as $\|x\|_p = (|x_1|^p + \cdots + |x_n|^p)^{1/p}$.
379 The L_∞-norm, or maximum norm, is defined as $\|x\|_\infty = \max(|x_1|, \ldots, |x_n|)$.

380 The result of Theorem 3.10 also holds for any L_p-norm. The key change that is
381 needed is in the proof of Lemma 3.7: For any two points $s, t \in [0,1)^d$, there exists
382 a shift v such that $s+v$ and $t+v$ are contained in a quadtree cell of side length at
383 most $2(D+1)\|s-t\|_p$. This extension follows easily from the proof of the Lemma (see
384 Appendix A.2). Theorem 3.10 then follows by adjusting ε by a factor of $2(D+1)d^{1/p}$ in
385 the last step, implying that the number of orderings will be $O(d^{1+1/p})^d(1/\varepsilon^d)\log(1/\varepsilon)$.
386 (For the L_∞-norm, ε only needs to be adjusted by a factor of $2(D+1)$.)

387 *Extension of locality-sensitive orderings for doubling metrics.* An abstraction of
388 low-dimensional Euclidean space, is a metric space with (low) doubling dimension.
389 Formally, a metric space $(\mathcal{M}, d)$ has *doubling dimension* λ if any ball of $\mathcal{M}$ of radius
390 r can be covered by at most 2^λ balls of half the radius (i.e., $r/2$). It is known that
391 $\mathbb{R}^d$ has doubling dimension $O(d)$ [37]. We point out that locality-sensitive orderings
392 still exist in this case, but they are less constructive in nature, since one needs to be
393 provided with all the points of interest in advance.

394 For a point set $P \subseteq \mathcal{M}$, the analogue of a quadtree for a metric space is a net tree
395 [21]. A net tree can be constructed as follows (the construction algorithm described

here is somewhat imprecise): The root node corresponds to the point set $P \subseteq \mathcal{M}$. Compute a randomized partition of P of diameter $1/2$ (assume P has diameter one), and for each cluster in the partition, create an associated node and hang it on the root. The tree is computed recursively in this manner, at each level i computing a random partition of diameter 2^{-i}. The leaves of the tree are points of P.

As with quadtrees, it is possible during this randomized construction for two nearby points to be placed in different clusters and be separated further down the tree. If $\ell = d(p,q)$ for two points $p, q \in P$, then the probability that p and q lie in different clusters of diameter $r = 2^{-i}$ in the randomized partition is at most $O((\ell/r)\log n)$ [15]. In particular, for $r \approx 1/(\ell \log n)$, the probability that p and q are separated is at most a constant. If we want this property to hold with high probability for all pairs of points, one needs to construct $O(\log n)$ (randomly constructed) net trees of P. (This corresponds to randomly shifting a quadtree $O(\log n)$ times in the Euclidean setting.)

Given such a net tree T, each node has $I = 2^{O(\lambda)}$ children. We can arbitrarily and explicitly number the children of each node by a distinct label from $[\![I]\!]$. One can define an ordering of such a tree as we did in the Euclidean case, except that the gap (in diameter) between a node and its children is $O(\varepsilon/\log n)$ instead of ε. Repeating our scheme in the Euclidean case, this implies that one would expect to require $(\varepsilon^{-1}\log n)^{O(\lambda)}$ orderings of P.

This requires having all the points of P in advance, which is a strong assumption for a dynamic data structure (as in some of the applications below). For example, Gottlieb and Roditty [19] show how to maintain dynamic spanners in a doubling metric, but only assuming that after a point has been deleted from P, the distance between the deleted point and a point currently in P can still be computed in constant time.

4. Applications.

4.1. Bichromatic closest pair.

Given an ordering $\sigma \in \Pi^+$, and two finite sets of points R, B in $\mathbb{R}^d$, let $\mathcal{Z} = \mathcal{Z}(\sigma, R, B)$ be the set of all pairs of points in $R \times B$ that are adjacent in the ordering of $R \cup B$ according to σ. Observe that inserting or deleting a single point from these two sets changes the contents of $\mathcal{Z}$ by a constant number of pairs. Furthermore, a point participates in at most two pairs.

LEMMA 4.1. *Let R and B be two sets of points in $[0,1)^d$, and let $\varepsilon \in (0,1)$ be a parameter. Let $\sigma \in \Pi^+$ be a locality-sensitive ordering (see Theorem 3.10). Then, one can maintain the set $\mathcal{Z} = \mathcal{Z}(\sigma, R, B)$ under insertions and deletions to R and B. In addition, one can maintain the closest pair in $\mathcal{Z}$ (under the Euclidean metric). Each update takes $O(d \log n \log(1/\varepsilon))$ time, where n is the total size of R and B during the update operation.*

Proof. Maintain two balanced binary search trees T_R and T_B storing the points in R and B, respectively, according to the order σ. Insertion, deletion, predecessor query and successor query can be implemented in $O(d\log(1/\varepsilon)\log n)$ time (since any query requires $O(\log n)$ comparisons each costing $O(d\log(1/\varepsilon))$ time by Lemma 3.9). We also maintain a min-heap of the pairs in $\mathcal{Z}$ sorted according to the Euclidean distance. The minimum is the desired closest pair. Notice that a single point can participate in at most two pairs in $\mathcal{Z}$.

We now explain how to handle updates. Given a newly inserted point r (say a red point that belongs to R), we compute the (potential) pairs it participates in, by computing its successor r' in R, and its successor b' in B. If $r \prec_\sigma b' \prec_\sigma r'$ then the new pair rb' should be added to $\mathcal{Z}$. The pair before r in the ordering that might use

r is computed in a similar fashion. In addition, we recompute the predecessor and successor of r in R, and we recompute the pairs they might participate in (deleting potentially old pairs that are no longer valid).

Deletion is handled in a similar fashion—all points included in pairs with the deleted point recompute their pairs. In addition, the successor and predecessor (of the same color) need to recompute their pairs. This all requires a constant number of queries in the two trees, and thus takes the running time as stated. □

THEOREM 4.2. *Let R and B be two sets of points in $[0,1)^d$, and let $\varepsilon \in (0, 1/2]$ be a parameter. Then one can maintain a $(1+\varepsilon)$-approximation to the bichromatic closest pair in $R \times B$ under updates (i.e., insertions and deletions) in $O_d(\log n \log^2(1/\varepsilon)/\varepsilon^d)$ time per operation, where n is the total number of points in the two sets. The data structure uses $O_d(n \log(1/\varepsilon)/\varepsilon^d)$ space, and at all times maintains a pair of points $r \in R$, $b \in B$, such that $\|r - b\| \leq (1+\varepsilon)d(R,B)$, where $d(R,B) = \min_{r \in R, b \in B} \|r - b\|$.*

Proof. We maintain the data structure of Lemma 4.1 for all the locality-sensitive orderings of Theorem 3.10. All the good pairs for these data structures can be maintained together in one global min-heap. The claim is that the minimum length pair in this heap is the desired approximation.

To see that, consider the bichromatic closest pair $r \in R$ and $b \in B$. By Theorem 3.10 there is a locality-sensitive ordering σ, such that the interval I in the ordering between r and b contains points that are in distance at most $\ell = \varepsilon \|r - b\|$ from either r or b. In particular, let P_r (resp., P_b) be all the points in I in distance at most ℓ from r (resp., b). Observe that $P_r \subseteq R$, as otherwise, there would be a bichromatic pair in P_R, and since the diameter of this set is at most ℓ, this would imply that (r, b) is not the closest bichromatic pair—a contradiction. Similarly, $P_b \subseteq B$. As such, there must be two points $b' \in B$ and $r' \in R$, that are consecutive in σ, and this is one of the pairs considered by the algorithm (as it is stored in the min-heap). In particular, by the triangle inequality, we have

$$\|r' - b'\| \leq \|r' - r\| + \|r - b\| + \|b - b'\| \leq 2\ell + \|r - b\| \leq (1 + 2\varepsilon)\|r - b\|.$$

The theorem follows after adjusting ε by a factor of 2. □

Remark. In the word RAM model, for integer input in $\{1, \ldots, U\}^d$, the update time can be improved to $O_d((\log\log U)\log^2(1/\varepsilon)/\varepsilon^d)$ expected, by using van Emde Boas trees [36] in place of the binary search trees (and the min-heaps as well). With standard word operations, we may not be able to explicitly map each point to an integer in one dimension following each locality-sensitive ordering, but we can still simulate van Emde Boas trees on the input as if the mapping has been applied. Each recursive call in the van Emde Boas recursion focuses on a specific block of bits of each input coordinate value (after shifting); we can extract these blocks, and perform the needed hashing operations on the concatenation of these blocks over the d coordinates of each point.

4.2. Dynamic spanners.

DEFINITION 4.3. *For a set of n points P in $\mathbb{R}^d$ and a parameter $t \geq 1$, a* **t-spanner** *of P is an undirected graph $G = (P, E)$ such that for all $p, q \in P$,*

$$\|p - q\| \leq \mathsf{d}_G(p,q) \leq t\|p - q\|,$$

where $\mathsf{d}_G(p,q)$ is the length of the shortest path from p to q in G using the edge set E.

Using a small modification of the results in the previous section, we easily obtain a dynamic $(1+\varepsilon)$-spanner. Note that there is nothing special about how the data structure in Theorem 4.2 deals with the bichromatic point set. If the point set is monochromatic, modifying the data structure in Lemma 4.1 to account for the closest monochromatic pair of points leads to a data structure with the same bounds and maintains the $(1+\varepsilon)$-approximate closest pair.

The construction of the spanner is very simple: Given P and $\varepsilon \in (0,1)$, maintain orderings of the points specified by Π^+ (see Theorem 3.10). For each $\sigma \in \Pi^+$, let E_σ be the edge set consisting of edges connecting two consecutive points according to σ, with weight equal to their Euclidean distance. Thus $|E_\sigma| = n-1$. Our spanner $G = (P, E)$ then consists of the edge set $E = \bigcup_{\sigma\in\Pi^+} E_\sigma$.

THEOREM 4.4. *Let P be a set of n points in $[0,1)^d$ and $\varepsilon \in (0,1/2]$. One can compute a $(1+\varepsilon)$-spanner G of P with $O_d(n\log(1/\varepsilon)/\varepsilon^d)$ edges, where every vertex has degree $O_d(\log(1/\varepsilon)/\varepsilon^d)$. Furthermore, a point can be inserted or deleted in $O_d(\log n \log^2(1/\varepsilon)/\varepsilon^d)$ time, where each insertion or deletion creates or removes at most $O_d(\log(1/\varepsilon)/\varepsilon^d)$ edges in the spanner.*

Proof. The construction is described above. The same analysis as in the proof of Theorem 4.2 implies the number of edges in G and the update time.

It remains to prove that G is a spanner. By Theorem 3.10, for any pair of points $s, t \in P$, there is a locality-sensitive ordering $\sigma \in \Pi^+$, such that the σ-interval $[s,t)$ contains only points that are in distance at most $\varepsilon\|s-t\|$ from either s or t. In particular, there must be two points in $s', t' \in P$ that are adjacent in σ, such that one of them, say s' (resp., t') is in distance at most $\varepsilon\|s-t\|$ from s (resp., t). As such, the edge $s't'$ exists in the graph being maintained.

This property is already enough to imply that this graph is a $(1+c\varepsilon)$-spanner for a sufficiently large constant c—this follows by an induction on the distances between the points (specifically, in the above, we apply the induction hypothesis on the pairs s, s' and t, t'). We omit the easy but somewhat tedious argument—see [5] or [20, Theorem 3.12] for details. The theorem follows after adjusting ε by a factor of c. ∎

4.2.1. Static and dynamic vertex-fault-tolerant spanners.

DEFINITION 4.5. *For a set of n points P in $\mathbb{R}^d$ and a parameter $t \geq 1$, a* ***k-vertex-fault-tolerant t-spanner*** *of P, denoted by (k,t)-VFTS, is a graph $G = (P, E)$ such that*

(i) G is a t-spanner (see Definition 4.3), and

(ii) For any $P' \subseteq P$ of size at most k, the graph $G \setminus P'$ is a t-spanner for $P \setminus P'$.

A $(k, 1+\varepsilon)$-VFTS can be obtained by modifying the construction of the $(1+\varepsilon)$-spanner in Section 4.2. Construct a set of locality-sensitive orderings Π^+. For each $\sigma \in \Pi^+$ and each $p \in P$, connect p to its $k+1$ successors and $k+1$ predecessors according to σ with edge weights equal to the Euclidean distances. Thus each ordering maintains $O(nk)$ edges and there are $O(|\Pi^+|\,kn) = O_d(kn\log(1/\varepsilon)/\varepsilon^d)$ edges overall. We now prove that this graph G is in fact a $(k, 1+\varepsilon)$-VFTS.

THEOREM 4.6. *Let P be a set of n points in $[0,1)^d$ and $\varepsilon \in (0,1/2]$. One can compute a k-vertex-fault-tolerant $(1+\varepsilon)$-spanner G for P in time*

$$O_d\big((n\log n\log(1/\varepsilon) + kn)\log(1/\varepsilon)/\varepsilon^d\big).$$

The number of edges is $O_d(kn\log(1/\varepsilon)/\varepsilon^d)$ and the maximum degree is bounded by $O_d(k\log(1/\varepsilon)/\varepsilon^d)$.

Furthermore, one can maintain the k-vertex-fault-tolerant $(1+\varepsilon)$-spanner G under insertions and deletions of points in $O_d\big((\log n \log(1/\varepsilon) + k)\log(1/\varepsilon)/\varepsilon^d\big)$ time per operation.

Proof. The construction algorithm, number of edges, and maximum degree follows from the discussion above. So, consider deleting a set $P' \subseteq P$ of size at most k from G. Consider an ordering $\sigma \in \Pi^+$ with the points P' removed. By the construction of G, all the pairs of points of $P \setminus P'$ that are (now) adjacent in σ remain connected by an edge in $G \setminus P'$. The argument of Theorem 4.4 implies that the remaining graph is spanner. We conclude that $G \setminus P'$ is a $(1+\varepsilon)$-spanner for $P \setminus P'$.

As for the time taken to handle insertions and deletions, one simply maintains the orderings of the points using balanced search trees. After an insertion of a point to one of the orderings in $O(\log n \log(1/\varepsilon))$ time, $O(k)$ edges have to be added and deleted. Therefore inserting a point takes $O\big((\log n \log(1/\varepsilon) + k)\,|\Pi^+|\big) = O_d\big((\log n \log(1/\varepsilon) + k)\log(1/\varepsilon)/\varepsilon^d\big)$ time total. Deletions are handled similarly.

The total construction time follows by inserting each of the points into the dynamic data structure. ∎

4.3. Dynamic approximate nearest neighbors. Another application of the same data structure in Theorem 4.2 is supporting $(1+\varepsilon)$-approximate nearest neighbor queries. In this scenario, the data structure must support insertions and deletions of points and the following queries: given a point q, return a point $t \in P$ such that $\|q-t\| \leq (1+\varepsilon)\min_{p\in P}\|q-p\|$.

THEOREM 4.7. *Let P be a set of n points in $[0,1)^d$. For a given $\varepsilon \in (0,1/2]$, one can build a data structure using $O_d(n\log(1/\varepsilon)/\varepsilon^d)$ space, that supports insertion and deletion in time $O_d(\log n \log^2(1/\varepsilon)/\varepsilon^d)$. Furthermore, given a query point $q \in [0,1)^d$, the data structure returns a $(1+\varepsilon)$-approximate nearest neighbor in P in $O_d(\log n \log^2(1/\varepsilon)/\varepsilon^d)$ time.*

Proof. Maintain the data structure of Lemma 4.1 for all locality-sensitive orderings of Theorem 3.10, with one difference: Since the input is monochromatic, for each locality-sensitive ordering $\sigma \in \Pi^+$, we store the points in a balanced binary search tree according to σ. The space and update time bounds easily follow by the same analysis.

Given a query point $q \in [0,1)^d$, for each of the orderings the algorithm inspects the predecessor and successor to q. The algorithm returns the closest point to q encountered. We claim that the returned point p is the desired approximate nearest neighbor.

Let $p^\star \in P$ be the nearest neighbor to q and $\ell = \|q-p^\star\|$. By Theorem 3.10, there is a locality-sensitive ordering $\sigma \in \Pi^+$ such that the σ-interval $I = [p^\star, q)$ contains points that are of distance at most $\varepsilon\ell$ from $p^\star$ or q (and this interval contains at least one point of P, namely, $p^\star$). Note that no point of P can be at distance less than $\varepsilon\ell$ to q. Thus, the point $p \in P$ adjacent to q in I is of distance at most $\varepsilon\ell$ from $p^\star$. Therefore, for such a point p, we have $\|p-q\| \leq \|p-p^\star\| + \|p^\star - q\| \leq (1+\varepsilon)\ell$.

The final query time follows from the time taken for these predecessor and successor queries, as in the proof of Lemma 4.1. ∎

5. Conclusion. In this paper, we showed that any bounded subset of $\mathbb{R}^d$ has a collection of "few" orderings which captures proximity. This readily leads to simplified and improved approximate dynamic data structures for many fundamental proximity-based problems in computational geometry. Beyond these improvements, we believe that the new technique could potentially be simple enough to be useful in practice,

and could be easily taught in an undergraduate level class (replacing, for example, well-separated pair decomposition—a topic that is not as easily accessible).

We expect other applications to follow from the technique presented in this paper. For example, recently Buchin et al. [4] presented a near linear-sized construction for robust spanners. The idea is to build a robust spanner in one dimension, and then obtain a robust spanner in higher dimensions by applying the one-dimensional construction using the locality-sensitive orderings.

Acknowledgments.. The authors thank the anonymous referees for their detailed and useful comments.

REFERENCES

[1] B. Alspach, *The wonderful Walecki construction*, Bull. Inst. Combin. Appl., 52 (2008), pp. 7–20.

[2] S. Arora, *Polynomial time approximation schemes for Euclidean TSP and other geometric problems*, J. Assoc. Comput. Mach., 45 (1998), pp. 753–782, https://doi.org/10.1109/sfcs.1996.548458.

[3] M. W. Bern, *Approximate closest-point queries in high dimensions*, Inform. Process. Lett., 45 (1993), pp. 95–99, https://doi.org/10.1016/0020-0190(93)90222-U.

[4] K. Buchin, S. Har-Peled, and D. Oláh, *A spanner for the day after*, in Proc. 35th Int. Annu. Sympos. Comput. Geom. *(SoCG)*, vol. 129 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019, pp. 19:1–19:15, https://doi.org/10.4230/LIPIcs.SoCG.2019.19.

[5] P. B. Callahan and S. R. Kosaraju, *Faster algorithms for some geometric graph problems in higher dimensions*, in Proc. 4th ACM-SIAM Sympos. Discrete Alg. *(SODA)*, V. Ramachandran, ed., ACM/SIAM, 1993, pp. 291–300.

[6] T. H. Chan, M. Li, L. Ning, and S. Solomon, *New doubling spanners: Better and simpler*, SIAM J. Comput., 44 (2015), pp. 37–53, https://doi.org/10.1137/130930984.

[7] T. M. Chan, *Approximate nearest neighbor queries revisited*, Discrete Comput. Geom., 20 (1998), pp. 359–373, https://doi.org/10.1007/PL00009390.

[8] T. M. Chan, *Closest-point problems simplified on the RAM*, in Proc. 13th ACM-SIAM Sympos. Discrete Alg. *(SODA)*, SIAM, 2002, pp. 472–473, http://dl.acm.org/citation.cfm?id=545381.545444.

[9] T. M. Chan, *A minimalist's implementation of an approximate nearest neighbor algorithm in fixed dimensions*. http://tmc.web.engr.illinois.edu/sss.ps, 2006.

[10] T. M. Chan, *Well-separated pair decomposition in linear time?*, Inform. Process. Lett., 107 (2008), pp. 138–141, https://doi.org/10.1016/j.ipl.2008.02.008.

[11] T. M. Chan, S. Har-Peled, and M. Jones, *On locality-sensitive orderings and their applications*, in 10th Innovations in Theo. Comput. Sci. *(ITCS)*, 2019, https://doi.org/10.4230/LIPIcs.ITCS.2019.21.

[12] T. M. Chan and D. Skrepetos, *Dynamic data structures for approximate Hausdorff distance in the word RAM*, Comput. Geom. Theory Appl., 60 (2017), pp. 37–44, https://doi.org/10.1016/j.comgeo.2016.08.002.

[13] A. Czumaj and H. Zhao, *Fault-tolerant geometric spanners*, Discrete Comput. Geom., 32 (2004), pp. 207–230, https://doi.org/10.1007/s00454-004-1121-7.

[14] D. Eppstein, *Dynamic Euclidean minimum spanning trees and extrema of binary functions*, Discrete Comput. Geom., 13 (1995), pp. 111–122, https://doi.org/10.1007/bf02574030.

[15] J. Fakcharoenphol, S. Rao, and K. Talwar, *A tight bound on approximating arbitrary metrics by tree metrics*, J. Comput. Sys. Sci., 69 (2004), pp. 485–497, https://doi.org/10.1016/j.jcss.2004.04.011.

[16] U. Feige and R. Krauthgamer, *Stereoscopic families of permutations, and their applications*, in Proc. 5th Israel Symp. Theo. Comput. and Systems *(ISTCS)*, IEEE Computer Society, 1997, pp. 85–95, https://doi.org/10.1109/ISTCS.1997.595160.

[17] M. L. Fredman and D. E. Willard, *Surpassing the information theoretic bound with fusion trees*, J. Comput. Sys. Sci., 47 (1993), pp. 424–436, https://doi.org/10.1016/0022-0000(93)90040-4.

[18] L. Gottlieb and L. Roditty, *Improved algorithms for fully dynamic geometric spanners and geometric routing*, in Proc. 19th ACM-SIAM Sympos. Discrete Alg. *(SODA)*, 2008, pp. 591–600, http://dl.acm.org/citation.cfm?id=1347082.1347148.

[19] L. Gottlieb and L. Roditty, *An optimal dynamic spanner for doubling metric spaces*, in Proc. 16th Annu. Euro. Sympos. Alg. *(ESA)*, 2008, pp. 478–489, https://doi.org/10.1007/978-3-540-87744-8_40.
[20] S. Har-Peled, *Geometric Approximation Algorithms*, vol. 173 of Math. Surveys & Monographs, Amer. Math. Soc., Boston, MA, USA, 2011, https://doi.org/10.1090/surv/173.
[21] S. Har-Peled and M. Mendel, *Fast construction of nets in low dimensional metrics, and their applications*, SIAM J. Comput., 35 (2006), pp. 1148–1184, https://doi.org/10.1137/S0097539704446281.
[22] D. S. Hochbaum and W. Maass, *Approximation schemes for covering and packing problems in image processing and VLSI*, J. Assoc. Comput. Mach., 32 (1985), pp. 130–136, https://doi.org/10.1145/2455.214106.
[23] J. Holm, E. Rotenberg, and C. Wulff-Nilsen, *Faster fully-dynamic minimum spanning forest*, in Proc. 23rd Annu. Euro. Sympos. Alg. *(ESA)*, N. Bansal and I. Finocchi, eds., vol. 9294 of Lect. Notes in Comp. Sci., Springer, 2015, pp. 742–753, https://doi.org/10.1007/978-3-662-48350-3_62.
[24] P. Indyk and R. Motwani, *Approximate nearest neighbors: Towards removing the curse of dimensionality*, in Proc. 30th ACM Sympos. Theory Comput. *(STOC)*, 1998, pp. 604–613, https://doi.org/10.1145/276698.276876.
[25] I. Kamel and C. Faloutsos, *On packing R-trees*, in Proc. 2nd Intl. Conf. Info. Knowl. Mang., B. K. Bhargava, T. W. Finin, and Y. Yesha, eds., ACM, 1993, pp. 490–499, https://doi.org/10.1145/170088.170403.
[26] S. Kapoor and X. Li, *Efficient construction of spanners in d-dimensions*, CoRR, abs/1303.7217 (2013).
[27] C. Levcopoulos, G. Narasimhan, and M. H. M. Smid, *Efficient algorithms for constructing fault-tolerant geometric spanners*, in Proc. 30th ACM Sympos. Theory Comput. *(STOC)*, J. S. Vitter, ed., ACM, 1998, pp. 186–195, https://doi.org/10.1145/276698.276734.
[28] S. Liao, M. A. López, and S. T. Leutenegger, *High dimensional similarity search with space filling curves*, in Proc. 17th Int. Conf. on Data Eng. *(ICDE)*, 2001, pp. 615–622, https://doi.org/10.1109/ICDE.2001.914876.
[29] T. Lukovszki, *New results of fault tolerant geometric spanners*, in Proc. 6th Workshop Alg. Data Struct. *(WADS)*, F. K. H. A. Dehne, A. Gupta, J. Sack, and R. Tamassia, eds., vol. 1663 of Lect. Notes in Comp. Sci., Springer, 1999, pp. 193–204, https://doi.org/10.1007/3-540-48447-7_20.
[30] T. Lukovszki, *New results on geometric spanners and their applications*, PhD thesis, University of Paderborn, Germany, 1999.
[31] G. M. Morton, *A computer oriented geodetic data base and a new technique in file sequencing*, tech. report, IBM, Ottawa, Ontario, March 1966, https://domino.research.ibm.com/library/cyberdig.nsf/0/0dabf9473b9c86d48525779800566a39.
[32] G. Peano, *Sur une courbe, qui remplit toute une aire plane*, Mathematische Annalen, 36 (1890), pp. 157–160, https://doi.org/10.1007/BF01199438, https://doi.org/10.1007/BF01199438.
[33] L. Roditty, *Fully dynamic geometric spanners*, Algorithmica, 62 (2012), pp. 1073–1087, https://doi.org/10.1007/s00453-011-9504-7.
[34] H. Sagan, *Space-filling curves*, Universitext Series, Springer-Verlag, 1994, https://doi.org/10.1007/978-1-4612-0871-6, https://books.google.com/books?id=gUfvAAAAMAAJ.
[35] S. Solomon, *From hierarchical partitions to hierarchical covers: Optimal fault-tolerant spanners for doubling metrics*, in Proc. 46th ACM Sympos. Theory Comput. *(STOC)*, D. B. Shmoys, ed., ACM, 2014, pp. 363–372, https://doi.org/10.1145/2591796.2591864.
[36] P. van Emde Boas, *Preserving order in a forest in less than logarithmic time and linear space*, Inf. Process. Lett., 6 (1977), pp. 80–82, https://doi.org/10.1016/0020-0190(77)90031-X.
[37] J.-L. Verger-Gaugry, *Covering a ball with smaller equal balls in $\mathbb{R}^n$*, Discrete Comput. Geom., 33 (2005), pp. 143–155, https://doi.org/10.1007/s00454-004-2916-2.

Appendix A. Proofs.

A.1. Proof of Lemma 3.1. Restatement of Lemma 3.1.

For n elements $\{0,\ldots,n-1\}$, there is a set $\mathfrak{O}$ of $\lceil n/2\rceil$ orderings of the elements, such that, for all $i,j\in\{0,\ldots,n-1\}$, there exists an ordering $\sigma\in\mathfrak{O}$ in which i and j are adjacent.

Proof. As mentioned earlier this is well known [1]. Assume n is even, and consider the clique K_n, with its vertices $v_0,\ldots,v_{n-1}$. The edges of this clique can be covered

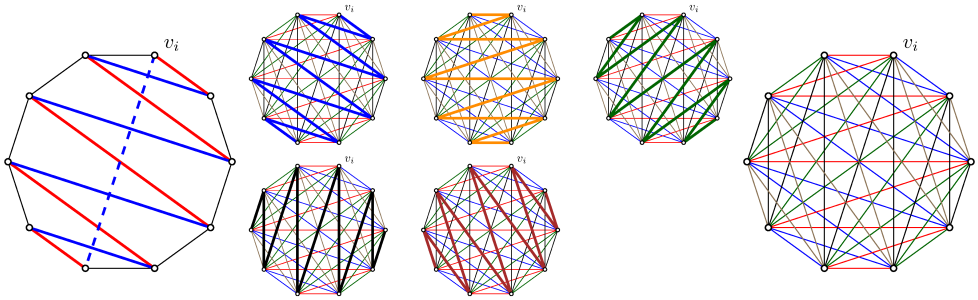

Figure A.1: For n even, a decomposition of K_n into $n/2$ Hamiltonian paths.

by $n/2$ Hamiltonian paths that are edge disjoint. Tracing one of these path gives rise to one ordering, and doing this for all paths results with orderings with the desired property, since edge v_iv_j is adjacent in one of these paths.

To get this cover, draw K_n by using the vertices of an n-regular polygon, and draw all the edges of K_n as straight segments. For every edge v_iv_{i+1} of K_n there are exactly $n/2$ parallel edges with this slope (which form a matching). Let M_i denote this matching. Similarly, for the vertex v_i, consider the segment $v_iv_{i+n/2}$ (indices are here modulo n), and the family of segments (i.e., edges) of K_n that are orthogonal to this segment. This family is also a matching M_i' of size $n/2-1$. Observe that $\sigma_i = M_i \cup M_i'$ forms a Hamiltonian path, as shown in Figure A.1. Since the slopes of the segments in M_i and M_i' are unique, for $i = 0, \ldots, n/2-1$, it follows that $\sigma_0, \ldots, \sigma_{n/2-1}$ are an edge-disjoint cover of all the edges of K_n by $n/2$ Hamiltonian paths.

If n is odd, use the above construction for $n+1$, and delete the redundant symbol from the computed orderings. ∎

A.2. Proof of Lemma 3.7 (shifting). For two positive real numbers x and y, let

$$x \,\%\, y = x - y\lfloor x/y \rfloor .$$

The basic idea behind shifting is that one can pick a set of values that look the "same" in all resolutions.

LEMMA A.1. *Let $n > 1$ be a positive odd integer, and consider the set $X = \{i/n \mid i = 0, \ldots, n-1\}$. Then, for any $\alpha = 2^{-\ell}$, where $\ell \geq 0$ is integer, we have that $X \,\%\, \alpha = \{i/n \,\%\, \alpha \mid i = 0, \ldots, n-1\}$ is equal to the set $\alpha X = \{\alpha i/n \mid i = 0, \ldots, n-1\}$.*

Proof. The proof is by induction. For $\ell = 0$ the claim clearly holds. Next, assume the claim holds for some $i \geq 0$, and consider $\ell = i+1$. Setting $m = (n-1)/2$ and $\Delta = 2^{-i}/n$, we have by induction (and rearrangement) that

$$\begin{aligned} X \,\%\, 2^{-i} &= 2^{-i}X = \{0, \Delta, \ldots, 2m\Delta\} \\ &= \{0, (m+1)\Delta, \Delta, (m+2)\Delta, 2\Delta, \ldots, (m+m)\Delta, m\Delta\} . \end{aligned}$$

Setting $\delta = \Delta/2 = 2^{-i-1}/n$, we have

$$\begin{aligned}
X \,\%\, 2^{-i-1} &= \left(X \,\%\, 2^{-i}\right) \%\, 2^{-i-1} \\
&= \{0, (m+1)\Delta, \Delta, (m+2)\Delta, 2\Delta, \ldots, (m+j)\Delta,\ j\Delta,\ \ldots, (m+m)\Delta, m\Delta\} \,\%\, 2^{-i-1} \\
&= \{0, 2(m+1)\delta, 2\delta, 2(m+2)\delta, 4\delta, \ldots, 2(m+j)\delta,\ 2j\delta, \ldots, 2(m+m)\delta, 2m\delta\} \,\%\, 2^{-i-1} \\
&= \{0, \quad \delta, 2\delta, \quad 3\delta, 4\delta, \ldots,\ (2j-1)\delta,\ 2j\delta,\ \ldots, (2m-1)\delta, 2m\delta\},
\end{aligned}$$

since $(2m+1)\delta = n\delta = 2^{-i-1}$ and $2(m+j)\delta \,\%\, 2^{-i-1} = (2m+1+2j-1)\delta \,\%\, 2^{-i-1} = (2j-1)\delta$, for $j = 1, \ldots, m$. ∎

Restatement of Lemma 3.7. *Consider any two points $p, q \in [0,1)^d$, and let $\mathcal{T}$ be the infinite quadtree of $[0,2)^d$. For $D = 2\lceil d/2 \rceil$ and $i = 0, \ldots, D$, let $v_i = (i/(D+1), \ldots, i/(D+1))$. Then there exists an $i \in \{0, \ldots, D\}$, such that $p + v_i$ and $q + v_i$ are contained in a cell of $\mathcal{T}$ with side length $\leq 2(D+1)\|p-q\|$.*

Proof. We start with the assumption that d is even (this assumption will be removed at the end of the proof). Let $\ell \in \mathbb{N}$, such that for $\alpha = 2^{-\ell}$, we have

$$(d+1)\|p-q\| < \alpha \leq 2(d+1)\|p-q\|.$$

For $\tau \in [0,1]$, let $\mathsf{G} + \tau$ denote the (infinite) grid with side length α shifted by the point $(\tau, \ldots, \tau)$.

Let $X = \{i/(d+1) \mid i = 0, \ldots, d\}$ be the set of shifts considered. Since we are shifting a grid with side length α, the shifting is periodical with value α. It is thus sufficient to consider the shifts modulo α.

Let $p = (p_1, \ldots, p_d)$ and $q = (q_1, \ldots, q_d)$. Assume that $p_1 \leq q_1$. A shift τ is bad, for the first coordinate, if there is an integer i, such that $p_1 \leq \tau + i\alpha \leq q_1$. The set of bad shifts in the interval $[0, \alpha]$ is

$$B_1 = \left\{(p_1, q_1) + i\alpha \;\middle|\; i \in \mathbb{Z}\right\} \cap [0, \alpha].$$

The set B_1 is either an interval of length $|p_1 - q_1| \leq \|p-q\| < \alpha/(d+1)$, or two intervals (of the same total length) adjacent to 0 and α. In either case, B_1 can contain at most one point of $\alpha X = X \,\%\, \alpha$, since the distance between any two values of αX is at least $\alpha/(d+1)$, by Lemma A.1.

Thus, the first coordinate rules out at most one candidate shift in $X \,\%\, \alpha$. Repeating the above argument for all d coordinates, we conclude that there is at least one shift in αX that is good for all coordinates. Let $\beta = \alpha i/(d+1) \in \alpha X$ this be good shift. Namely, p and q belong to the same cell of $\mathsf{G} + \beta$. The final step is to observe that shifting the points by $-\beta$, instead of the grid by distance β has the same effect (and $-\beta \,\%\, \alpha \in \alpha X$), and as such, the canonical cell containing both p and q is in the quadtree $\mathcal{T}$ as desired, and the side length of this cell is α.

Finally, if d is odd, replace d by $d+1$ in the above proof. This results in a set of $d + 2 = 2\lceil d/2 \rceil + 1 = D + 1$ shifts. ∎