Navigating nets: Simple algorithms for proximity search

[Extended Abstract]

Robert Krauthgamer *　　James R. Lee †

Abstract

We present a simple deterministic data structure for maintaining a set S of points in a general metric space, while supporting proximity search (nearest neighbor and range queries) and updates to S (insertions and deletions). Our data structure consists of a sequence of progressively finer ϵ-nets of S, with pointers that allow us to navigate easily from one scale to the next.

We analyze the worst-case complexity of this data structure in terms of the "abstract dimensionality" of the metric S. Our data structure is extremely efficient for metrics of bounded dimension and is essentially optimal in a certain model of distance computation. Finally, as a special case, our approach improves over one recently devised by Karger and Ruhl [KR02].

1 Introduction

Nearest-neighbor search (NNS) is the problem of preprocessing a set S of n points lying in a huge (possibly infinite) metric space (M, d) so that given a query $q \in M$, one can efficiently locate the nearest point to q among the points in S.[1] Computing such nearest neighbors efficiently is a classical and fundamental problem with numerous practical applications. These include data compression, database queries, machine learning, computational biology, data mining, pattern recognition, and ad-hoc networks. A common feature of many of these examples is that comparing two elements is costly, hence the number of distance computations should be made as small as possible.

The general NNS problem has several flavors. For instance, the host metric space (M, d) may be application specific (for example, it could be the weighted edit distance between strings in a spell-checker or in genomic data), it may be infinite (e.g., Euclidean space), and it may even be so unstructured that it is practically unknown (as in a peer-to-peer network, representing the internode latencies). The space and time of the preprocessing stage may be constrained (e.g., to be polynomial or linear in n). The application at hand may require the data structure to be *dynamic* in the sense that it should efficiently support the insertion and deletion of points from S. Additionally, it may be desirable for the data structure to be distributed among the data points, in which case some other cost measures like communication and load could be highly important. Finally, it may be the case that only *approximate* solutions are required. Let q be the query and $a \in S$ the closest point to q in S. Then a $(1+\epsilon)$-NNS algorithm is one which, given q, returns some $s \in S$ such that $d(q, s) \leq (1+\epsilon)\, d(q, a)$.

Most previous research has focused on the important special case when $M = \mathbb{R}^d$ and distances are computed according to some ℓ_p norm. While many types of data can be naturally represented in such a form, this is certainly not true for a significant number of applications, and it is therefore desirable to address NNS in general metric spaces. On the one hand, data structures for general metrics might perform a nearest neighbor query in time as poorly as $\Omega(n)$ which is unacceptable in practice. Such a dependence is inherent even when only approximate solutions are required. A well-known example is where S forms a uniform metric, so that the interpoint distances in S are all equal, providing essentially no information. (See Section 3 for a specification of the model and additional lower bounds.) On the other hand, metrics arising in practice often have a nicer structure which can be exploited algorithmically.

Given this state of affairs, an increasing amount of recent attention has focused on understanding the complexity of NNS in terms of a metric's implicit structure. In Euclidean spaces, a natural and common measure for the metric's complexity is the dimension of the Euclidean host space. Indeed, the efficiency of most algorithms depend on this dimension. In fact, the

*IBM Almaden Research Center, 650 Harry Road, San Jose CA 95120, USA. Part of this work was done while this author was with the International Computer Science Institute and with the Computer Science Division of U.C. Berkeley. Email: robi@almaden.ibm.com

†Computer Science Division, U.C. Berkeley, Berkeley, CA 94720, USA. Part of this work was done while the author was at Microsoft Research (Redmond). Work partially supported by NSF grant CCR-0121555 and an NSF Graduate Research Fellowship. Email: jrl@cs.berkeley.edu

[1] The distance function d is assumed to be nonnegative and symmetric, and to satisfy the triangle inequality.

running time of many algorithms grows exponentially with the dimension, a phenomenon called the *curse of dimensionality* (notable exceptions are those of [IM98, KOR98]). It is thus only natural to try and define an analogous notion of *dimensionality* for abstract metric spaces.

Our first contribution is conceptual—we put forward (in the context of NNS) a natural notion of abstract dimension, which is based on one of Assouad [Ass83] (from the theory of analysis on metric spaces). This is corroborated by a technical contribution—we provide an efficient and dynamic data structure for NNS in general low-dimensional metrics. Other notions of dimensionality were suggested and studied in the context of NNS in general metric spaces [Cla99, KR02], and our scheme compares favorably with them. Furthermore, our scheme is comparable to the best one known for low-dimensional Euclidean space (with linear preprocessing space) [AMN$^+$98], despite the fact that our approach is general and, in particular, independent of the Euclidean geometry! See Section 1.3 for a more detailed account.

1.1 Abstract dimension. The *doubling dimension* of a metric space (X, d), denoted in this paper by $\dim(X)$, is the minimum value ρ such that every set in X can be covered by 2^ρ sets of half the diameter. (The *diameter* of a set $S \subseteq X$ is $\sup\{d(x, y) : x, y \in S\}$.) As usual, define the *(closed) ball of radius r about x in $S \subseteq X$* to be $B_S(x, r) = \{y \in S : d(x, y) \leq r\}$; we may omit the subscript S when it is clear from the context. It is not difficult to see that the minimum value ρ such that every *ball* in X can be covered by 2^ρ *balls* of half the *radius* is within a factor of two of the above definition—this is the value we work with throughout.

A metric space is called *doubling* if its dimension is $O(1)$. For example, a uniform metric on k points has dimension $\log k$. An approximate converse also holds (see Lemma 1.2 below), showing that small doubling dimension quantifies the lack of a large nearly-uniform metric. In other words, this notion of dimension measures the "volume growth" of X. We note that Clarkson [Cla99] used a similar notion (under a different name) in the context of NNS, but his results do not seem to unleash its full power. (See Section 1.3 for further details.)

The doubling dimension has several natural and appealing properties. Let (X, d) be an arbitrary metric space; here are some of the more simple properties (see, e.g. [Hei01]).

1. For $X = \mathbb{R}^k$ equipped with any norm, $\dim(X) = \Theta(k)$.
2. If X is a subspace of Y, then $\dim(X) \leq \dim(Y)$.
3. $\dim(X_1 \cup \cdots \cup X_m) \leq \max_i \dim(X_i) + \log m$.

It has been noted in [KR02] that doubling metrics (those of uniformly bounded dimension) occur naturally in practical applications like peer-to-peer networks. In feature recognition problems, the data set S is often contained in the union of m low-dimensional manifolds lying in some very high-dimensional space $\mathbb{R}^L$ and the distance function used is a norm of $\mathbb{R}^L$ (usually the ℓ_1 or ℓ_2 norm). For instance, a manifold may represent the feature vectors corresponding to different viewpoints of a single object, thus m objects give rise to a union of m manifolds. In this case, performing nearest neighbor search using only the structure of the high-dimensional host space would prove quite costly. The situation is made even more complicated by the fact that noise and errors arising in measurement actually yield a union of m sets each of which is only *close* to a manifold. Fortunately, the doubling dimension is fairly insensitive to such small perturbations.

1.2 Our results. We provide a simple scheme for $(1+\epsilon)$-NNS in general metrics. The data structure, described in Section 2, is deterministic and thus guaranteed to answer $(1+\epsilon)$-NNS queries correctly. Our data structure is *dynamic*; it supports insertions and deletions to S. It also supports range queries: One is given a query $q \in M$ and a range t, and the goal is to return all points $s \in S$ such that $d(q, s) \leq t$.

The complexity of these operations depends on the dimension of S and the *aspect ratio* of (S, d), denoted $\Delta = \Delta_S$, which is defined as the ratio between the largest and smallest interpoint distances in S. In most applications, $\Delta = \text{poly}(|S|)$, hence $\log \Delta = O(\log n)$.

If S is doubling, the data structure uses $O(n)$ space; it answers $(1+\epsilon)$-NNS queries in time $O(\log \Delta) + (1/\epsilon)^{O(1)}$; and it implements insertions and deletions in time $O(\log \Delta \log \log \Delta)$. The running time depends exponentially on $\dim(S)$ (see Section 2), but in a distance-oracle model, where access to M is restricted to queries to the distance function d, this dependence is necessary, as we show in Section 3.

In [KR02], a different notion of dimension is defined (implicitly) which we denote by $\dim_{\text{KR}}(X)$. This is the least value k such that $|B_X(x, 2r)| \leq 2^k |B_X(x, r)|$ for every point $x \in X$ and every $r > 0$. The following easy lemma is proved in [GKL03]; for completeness, we repeat the proof in the Appendix.

Lemma 1.1. ([GKL03]) *Every finite metric (X, d) satisfies $\dim(X) \leq 4 \dim_{\text{KR}}(X)$.*

The converse direction (a bound on $\dim_{\text{KR}}(X)$ in terms of $\dim(X)$) does not hold. In other words, doubling metrics (those of bounded doubling dimension) form a *strictly larger* class than those metrics of bounded KR-dimension. The two notions are further compared in the next section.

When the metric formed by the data set S together with the query point q (namely, $S \cup \{q\}$) happens to belong to the more restricted class of metrics with bounded KR-dimension, by a slight modification of our querying procedure (but not the data structure), we are able to return *exact* nearest-neighbors and to match or improve the bounds of [KR02]; see Section 1.3.

1.3 Related work. Several works (e.g., [Bri95]) demonstrate that the (concentration of the) histogram of distances between points in a metric space indicates (heuristically) its dimensionality. Chávez et al. [CNBYM01] suggest to define the dimension of a metric as $\rho = \mu^2/2\sigma^2$ where μ and σ^2 are the histogram's mean and variance, and show that it affects, in a random instance, the efficiency of a certain nearest neighbor search algorithm. Faloutsos and Kamel [FK97] suggest to measure the fractal dimension of the data set, but this notion only applies to Euclidean spaces.

Clarkson [Cla99] devised two NNS data structures for metric spaces satisfying a certain sphere packing property which is equivalent to having $O(1)$ doubling dimension. (See Table 1.) However, his data structures are randomized, not dynamic, the query time is super-logarithmic, and it is assumed that the data set S and the query q are drawn from the same (unknown) probability distribution. Our algorithms improve over [Cla99] in these respects, although they only guarantee $(1+\epsilon)$-NNS.

More recently, Karger and Ruhl [KR02] suggested the notion of dimension discussed in the previous section. One justification for this notion is that the k-dimensional grid with the l_1 metric has KR-dimension $\Theta(k)$. Karger and Ruhl [KR02] show an efficient NNS data structure for metric spaces with bounded KR-dimension. (See Table 1.)

Lemma 1.1 shows that the doubling dimension of every metric space is (essentially) not larger than its KR-dimension. Therefore, all our results for doubling metrics immediately hold for metrics with bounded KR-dimension. In addition, we can show that when run on the latter family of metrics, our algorithms can be (slightly) modified so as to find *exact* NNS with running time that is similar to [KR02], but with linear (instead of near-linear) space. Our results improve over [KR02] in several respects. Besides extending to a larger family of metrics, they are deterministic and do not require any parameter, while those of [KR02] are randomized (Las-Vegas), and their performance depends on a correct setting of the dimension parameter.

We take a moment to discuss the frailty of the notion of KR-dimension. In particular, as was pointed out in [KR02], subsets of metrics of bounded KR-dimension do **not** always have bounded dimension themselves. In particular, there are subsets of the real line which have *unbounded* KR-dimension (while certainly every subset of $\mathbb{R}$ has bounded doubling dimension). The reason for this is quite simple; the algorithms of [KR02] use random sampling in order to find nearest neighbors, and thus impose a certain uniformity on the layout of points in the metric space. Thus it is unclear whether their approach could be extended to the larger class of doubling metrics. To see a simple example of this instability, consider the discrete annulus $S = \{x \in \mathbb{Z} : 2r > |x| > r\}$. It is not difficult to see that $\dim_{\mathrm{KR}}(S) = O(1)$ (uniformly, for any value of $r > 0$). On the other hand, $\dim_{\mathrm{KR}}(S \cup \{0\}) = \Omega(\log r)$, thus the addition of a single point to S can cause the KR-dimension to grow arbitrarily.

Perhaps most interestingly, our results are comparable to those of [AMN$^+$98] for bounded dimensional Euclidean metrics (which, of course, also have bounded doubling dimension); see Table 1. It is quite remarkable that similar running times are achievable using only a bound on the volume growth of point sets and not the geometry of Euclidean space.

1.4 Techniques. Let (X, d) be a metric space. We say that a subset $Y \subseteq X$ is an *ϵ-net* of X if it satisfies (1) For every $x, y \in Y$, $d(x, y) \geq \epsilon$ and (2) $X \subseteq \bigcup_{y \in Y} B(y, \epsilon)$. Such nets always exist for any $\epsilon > 0$. For finite metrics, they can be constructed greedily. For arbitrary metrics, proof of their existence is an easy application of Zorn's lemma. This classical notion of an ϵ-net, which is a standard tool in the study of metric spaces (see e.g. [Hei01]), should not be confused with a more recent notion (under the same name) from Computational Geometry.

The following lemma will be key to our analysis.

LEMMA 1.2. *Let (S, d) be a metric space, and let $Y \subseteq S$. If the aspect ratio of the metric induced on Y is at most α and $\alpha \geq 2$, then $|Y| \leq \alpha^{O(\dim(S))}$.*

Proof. Let $d_{\min} = \inf\{d(x, y) : x, y \in Y\}$ and $d_{\max} = \sup\{d(x, y) : x, y \in Y\}$ be the minimum and maximum interpoint distance in Y, respectively, and assume that $\alpha = \frac{d_{\max}}{d_{\min}} < \infty$. Notice that Y is contained in a ball of radius $2d_{\max} \leq 2\alpha d_{\min}$ in S (centered at any point of Y). Applying the definition of doubling dimension iteratively several times we get that this ball, and in particular Y, can be covered by $2^{\dim(S) \cdot O(\log \alpha)}$ balls of radius $d_{\min}/3$. Each of these balls can cover at

	Dimension	Space	NNS or $(1+\epsilon)$-NNS	Insert/Delete
[Cla99] *	$\dim(S) = O(1)$ †	$O(n \log \Delta)$	$O(\log^4 n \cdot \log \Delta)$	–
[Cla99] *	$\dim(S) = O(1)$ ‡	$n(\log n)^{O(\log\log \Delta)}$	$(\log n)^{O(\log\log \Delta)}$	–
This paper	$\dim(S) = O(1)$	$O(n)$	$O(\log \Delta) + (1/\epsilon)^{O(1)}$	$O(\log \Delta \log\log \Delta)$
[KR02] *	$\dim_{\mathrm{KR}}(S) = O(1)$	$O(n \log n)$	$O(\log n)$	$O(\log n \log\log n)$
This paper	$\dim_{\mathrm{KR}}(S) = O(1)$	$O(n)$	$O(\log n)$	$O(\log n \log\log n)$
[AMN+98]	$O(1)$ Euclidean	$O(n)$	$(1/\epsilon)^{O(1)} \log n$	$O(\log n)$

* Randomized (Las Vegas) data structure

† Assuming there is a training set for the query.

‡ Assuming the query comes from the same (unknown) distribution as the data set.

Table 1: Comparison of NNS schemes for general metrics.

most one point of Y (by definition of $d_{\min}$) and thus $|Y| \le 2^{\dim(S)\cdot O(\log \alpha)} \le \alpha^{O(\dim(S))}$. □

A simplified 3-NNS algorithm. Here is a simplified version of our data structure. Let (S, d) be the metric space which is queried against, and for the sake of this informal discussion, assume that the minimum interpoint distance in S is $\min\{d(x, y) : x, y \in S\} = 1$. In this case, the aspect ratio Δ is just the diameter of S. In what follows, for a subset $R \subseteq S$ and a point $x \in M$, we let $d(x, R) = \inf_{y\in R} d(x, y)$.

Let $k = \log \Delta$, and for $i = 0, 1, \dots, k$, let Y_i be a 2^i-net of S. Now, for every point $y \in Y_i$, suppose we have the set of points $L_{y,i} = \{z \in Y_{i-1} : d(y, z) \le \gamma 2^i\}$, for some constant γ to be specified later.

Notice that the set $L_{y,i}$ has minimum distance 2^{i-1} (since this is the minimum distance in Y_{i-1}) and maximum distance $\gamma 2^{i+1}$, hence its aspect ratio is bounded. When S is a doubling metric, Lemma 1.2 implies that $|L_{y,i}| = O(1)$ where the constant depends on the choice of γ. Also, Y_k contains only one (arbitrary) point which we denote by y_{top}.

Now, given a query point $q \in M$, we first set $y = y_{\mathrm{top}}$, and then iteratively for $i = k, k-1, \dots$ find the point in $L_{y,i} \subseteq Y_{i-1}$ that is closest to q, and set y to be this point (for the next iteration). If, at some stage, we reach a point where $d(q, L_{y,i}) > 3 \cdot 2^{i-1}$, then we stop and output y. Otherwise, we output the last value $y \in Y_0$.

First, notice that the running time of this algorithm is at most $O(\log \Delta)$, since finding the closest point to q among a list $L_{y,i}$ takes constant time (the lists are of size $O(1)$). Thus we need only argue that the output point y is a 3-approximation to the nearest neighbor $a \in S$, i.e. that $d(q, y) \le 3\, d(q, a)$.

To this end, let j be such that $d(q, y) \le 3 \cdot 2^j$, but $d(q, L_{y,j}) > 3 \cdot 2^{j-1}$, i.e. the step in which the distance does not decrease by a factor of 2. First, we argue that $d(a, L_{y,j}) \le 2^{j-1}$. In other words, the closest 2^{j-1}-net point to a is contained in $L_{y,j}$. Let $y^* \in Y_{j-1}$ be such that $d(a, y^*) \le 2^{j-1}$. We need to argue that $d(y^*, y) \le \gamma 2^j$; in this case, we will have $y^* \in L_{y,j}$. Since $d(q, y) \le 3 \cdot 2^j$, we have

$$\begin{aligned} d(y^*, y) &\le d(y^*, a) + d(a, y) \le 2^{j-1} + d(a, q) + d(q, y) \\ &\le 2^{j-1} + 2 \cdot d(q, y) \le 7 \cdot 2^j, \end{aligned}$$

thus choosing $\gamma = 7$ suffices.

This shows that the descension process "tracks" the closest net point to a. Now it is a simple matter to see that

$$3 \cdot 2^{j-1} < d(q, L_{y,j}) \le d(q, a) + d(a, L_{y,j}) \le d(q, a) + 2^{j-1}$$

so that $d(q, a) > 2^j$. Since $d(q, y) \le 3 \cdot 2^j$, it follows that y is a 3-approximate nearest neighbor. Similar arguments show that if we end with $y \in Y_0$ then y is actually the closest point to q. Later, we will see that, after obtaining an $O(1)$-approximate nearest neighbor, the $(1+\epsilon)$-NNS problem can be solved in time $O(1/\epsilon)^{O(1)}$.

The preceding simple algorithm illustrates the power of using nets at different scales to navigate through a metric space, roughly halving the distance from q to the closest point at each step. All the operations in our data structure are implemented inside this simple framework.

To overcome some technical limitations of the above scheme, the actual data structure presented in Section 2 is more involved. First, the above algorithm has to be extended to $(1+\epsilon)$-NNS, for any $\epsilon > 0$. This can be achieved by roughly $\log(1/\epsilon)$ more iterations of the same decision procedure, but at each iteration we now have to process more than just one point y. Secondly, in the presence of insertions and deletions, we cannot make the simplifying assumption that the minimum interpoint distance in S is 1, and thus we must find a way to keep track of the "relevant" scales. Finally, for a technical reason, the above data structure does not support efficient deletions and might require space $\Omega(n \log \Delta)$. The main remedy for these latter problems is to choose Y_i to be a 2^i-net in Y_{i-1} rather than in S.

2 Navigating nets

In this section we present a data structure that maintains a set S of points in a metric space, so as to support proximity queries and updates to S. The performance guarantees of this data structure (in terms of time and space) depend on the dimensionality of the data set S, as defined in Section 1. However, the data structure is deterministic, and is guaranteed to be correct for any metric space unconditionally. Neither the dimension of S nor ϵ (for $(1+\epsilon)$-NNS) needs to be known in advance.

We start by describing the data structure in Section 2.1, and analyzing its space requirement in Section 2.2. We then present the procedures for computing proximity queries in Sections 2.3 and 2.4, and those for updating S in Sections 2.5 and 2.6. Finally, we give improved bounds for KR metrics in Section 2.7.

We shall use the notation of Section 1, where (M, d) is the host metric space, S is the set of points to be maintained by the data structure, and $n = |S|$. Let $d_{\max} := \sup\{d(x, y) : x, y \in S\}$ and $d_{\min} := \inf\{d(x, y) : x, y \in S\}$ denote the maximum and minimum interpoint distance in S, respectively so that $\Delta := d_{\max}/d_{\min}$ is the aspect ratio of S.

2.1 The data structure. Let $\Gamma = \{2^i : i \in \mathbb{Z}\}$, and let us call every value $r \in \Gamma$ a *scale*. To simplify the exposition, we consider infinitely many scales, but it will be apparent that only $O(\log \Delta)$ of them are "relevant" and the remaining scales will be trivial.

For every $r \in \Gamma$, let Y_r be an r-net of $Y_{r/2}$. As the base case, we define $Y_r := S$ for all scales $r \le d_{\min}$. For every $r \in \Gamma$ and every $y \in Y_r$, the data structure stores a list of the nearby points to y among the $r/2$-net $Y_{r/2}$. This *scale r navigation list of y* is defined by

$$L_{y,r} := \{z \in Y_{r/2} : d(z, y) \le \gamma \cdot r\}, \tag{2.1}$$

where $\gamma > 0$ is a universal constant. We will see later that $\gamma \ge 4$ suffices for all the operations. While Y_r need not be an r-net of S, the following lemma shows that Y_r is a relaxed variant of an r-net of S.

Lemma 2.1. *For every scale r, we have:*
1. Covering: *$d(z, Y_r) < 2r$ for every $z \in S$.*
2. Packing: *$d(x, y) \ge r$ for every $x, y \in Y_r$.*

Proof. The covering property follows by induction. The base case is $r < d_{\min}$, for which $Y_r = S$ and the desired property is trivial. For the inductive step, assume the property holds for scale r, i.e., there exists $y \in Y_r$ such that $d(z, y) < 2r$. Since Y_{2r} is a $2r$-net of Y_r, we get that $d(z, Y_{2r}) \le d(z, y) + d(y, Y_{2r}) < 4r$.

The packing property follows directly from the fact that Y_r is an r-net of $Y_{r/2}$. ◻

The next lemma upper bounds the size of any navigation list. Its proof follows by observing that all the points in a list $L_{y,r}$ are points of $Y_{r/2} \subseteq S$, so the distance between every two of them is at least $r/2$, while they all lie in a ball of radius γr. Applying Lemma 1.2 yields the following.

Lemma 2.2. *The size of every navigation list is at most $2^{O(\dim(S))}$.*

Implementation. We now discuss the implementation of this data structure. First, the nets Y_r are not maintained explicitly, but rather follow implicitly from the lists $L_{y,r}$, i.e., Y_r is the set of all points $y \in S$ for which $L_{y,r}$ exists. Second, if S is nonempty, we have by Lemma 2.1 that for every scale $r > d_{\max}$, the net Y_r consists of the same single point, which we denote y_{top}. The data structure maintains this point y_{top} and the cutoff scale $r_{\max} := \min\{r \in \Gamma : \forall r' \ge r, |Y_{r'}| = 1\}$ for the sake of bootstrapping most of the operations. Third, for all scales $r \le d_{\min}$ the net Y_r is equal to S, so for scales $r \le d_{\min}/2$, every point $x \in S$ has a *trivial* list $L_{x,r} = \{x\}$. These trivial lists can be represented succinctly by storing, for every $x \in S$, a scale $r_x \in \Gamma$ below which all lists of x are trivial. For the sake of analysis, define $r_{\min} := \min\{r_x : x \in S\}$.

We can now upper bound the number of navigation lists that need to be stored for any point $x \in S$.

Lemma 2.3. *$r_{\max} = \Theta(d_{\max})$ and $r_{\min} = \Theta(d_{\min})$, so every point has $O(\log \Delta)$ non-trivial navigation lists.*

Proof. Using Lemma 2.1 it is easy to see that $r_{\max} = \Theta(d_{\max})$. It is straightforward from the definition (2.1) that $r_x \ge \Omega(d_{\min})$ for every $x \in S$. In addition, $Y_{r_{\min}/2} = S$ so by Lemma 2.1 $d_{\min} \ge r_{\min}/2$. We conclude that a list for scale r needs to be stored only if $\Omega(d_{\min}) \le r_x \le r \le r_{\max} \le O(d_{\max})$. The lemma follows. ◻

Combining Lemmas 2.2 and 2.3 yields an upper bound on the total space required for the data structure of $n \cdot 2^{O(\dim(X))} \log \Delta$. We improve over this in Section 2.2, abolishing the $\log \Delta$ factor via a more careful analysis.

When analyzing the performance of the data structure, we assume that the navigation lists are stored as follows. For each point $x \in S$, the non-trivial navigation lists of x are stored using, say, a balanced search tree, which requires linear space and implements find, insert and delete in logarithmic time. It then follows from Lemma 2.3 that, say, inserting a new navigation list for a point can be done in time $O(\log\log\Delta)$.

Remark. It is possible to speed up the $(1+\epsilon)$-NNS procedure by letting each navigation list $L_{x,r}$ contain not only pointers to points $y \in Y_{r/2}$, but also pointers to their corresponding navigation lists $L_{y,r/2}$. Since the latter navigation list might be trivial (and not stored explicitly), we shall actually store a pointer to the navigation list $L_{y,r'}$, where r' is the largest scale for which $r' \leq r/2$ and $L_{y,r'}$ is non-trivial. In order to update these pointers in the presence of insertions and deletions, they must be implemented as bidirectional pointers.

2.2 Space requirement. We now show that for doubling metrics S, the total space used by our data structure is $O(n)$.

Theorem 2.1. *The size of the data structure is $2^{O(\dim(S))} \cdot n$ words.*

Proof. Recall that trivial navigation lists $L_{x,r} = \{x\}$ are represented implicitly. Hence, the total space used by our implementation of the data structure is linear in the space required to represent all the non-trivial navigation lists. The size of each such list is at most $2^{O(\dim(S))}$ by Lemma 2.2, and thus it suffices to show that the number of non-trivial navigation lists is $O(n)$. For simplicity in what follows, we shall assume that γ is a power of 2.

We bound the number of non-trivial navigation lists by a charging argument against the points of S. Each such list is charged against a point in S as follows. A non-trivial list $L_{x,r}$ must contain at least one point $y \neq x$. By definition, $x, y \in Y_{r/2}$ and thus $d(x,y) \geq r/2$. Furthermore, $d(x,y) \leq \gamma r$, and thus $Y_{2\gamma r}$ cannot contain both x and y. We charge the list $L_{x,r}$ to the point $z \in \{x,y\}$ which is *not* contained in $Y_{2\gamma r}$.

It remains to bound the number of navigation lists that are charged to any single point $z \in S$. Consider a non-trivial scale r list that is charged against z. This list must contain z and also another point z', such that the list is either $L_{z,r}$ or $L_{z',r}$. Let $r^* = r^*(z)$ be the largest scale for which $z \in Y_{r^*}$. The charging scheme is such that $r^* \in \{r/2, r, \ldots, \gamma r\}$. It follows that once z is fixed, r can have only $O(1)$ distinct values. Furthermore, z, z' belong to the same scale r list, and thus $r/2 \leq d(z,z') \leq \gamma r$. It follows that once z and r are fixed, there are only $2^{O(\dim(S))}$ possible points z' (Lemma 1.2). As mentioned before, once the triple z, z', r is fixed, there are only two navigation lists ($L_{z,r}$ and $L_{z',r}$). We conclude that the number of lists charged to any single point z is $2^{O(\dim(S))}$, and the total number of non-trivial lists is indeed $2^{O(\dim(S))} n$. $\square$

2.3 Approximate nearest neighbor query. We now present a procedure that uses the above data structure to find a $(1+\epsilon)$-approximate nearest neighbor for a query point. That is, given a point $q \in M$ and a value $\epsilon > 0$, the procedure finds a point $p \in S$ with $d(q,p) < (1+\epsilon)\, d(q,S)$. We stress that ϵ is arbitrary, both in the sense that the data structure is oblivious to ϵ, and in the sense that ϵ need not be small; for instance, setting $\epsilon = 1$ we can find a 2-approximate nearest neighbor extremely fast. Choosing smaller ϵ trades speed for accuracy; the precise result is as follows.

Theorem 2.2. *A $(1+\epsilon)$-approximate nearest neighbor among S can be computed using the data structure in time $2^{O(\dim(S))} \log\Delta + (1/\epsilon)^{O(\dim(S))}$. This bounds, in particular, the number of distance computations.*

To prove this theorem we next present Procedure Approx-NNS. We prove its correctness in Lemma 2.5 and analyze the running time in Lemma 2.6. We shall make use of the following definition.

Definition. $Z_r \subseteq Y_r$ is called *non-proper* if it contains only one point, x, and $r_x > r$. Otherwise, Z_r is *proper*.

The algorithm. We now outline Procedure Approx-NNS; the full description is given in Figure 2.3. The procedure iterates over the nets Y_r, starting with the largest (non-trivial) scale $r = r_{\max}$ (line 1), and iteratively decreasing r by a factor of 2 (lines 2-4). At each iteration, the goal is to construct a subset $Z_{r/2} \subseteq Y_{r/2}$ containing the points of $Y_{r/2}$ that are nearby q, as formalized in Lemma 2.4. This is done efficiently by exploiting Z_r from the previous iteration and the navigation lists of the corresponding points (line 3). The crux is that Z_r then contains a point of S whose distance from q is at most $d(q,S) + r$. The iterations continue roughly until $r \leq \epsilon \cdot d(q,S)$ (line 2), and then we simply report the closest point to q among Z_r (line 5).

The analysis. By induction, $Z_r \subseteq Y_r$ for every set Z_r computed by this procedure, and thus $L_{z,r}$ in line 3 is well-defined.

Lemma 2.4. *Let a be a closest point to q among S. Then every set Z_r computed by Procedure* Approx-NNS *contains a point z_r with $d(a, z_r) \leq 2r$.*

Proof. Proceed by induction on r. For the base case $r = r_{\max}$, line 1 sets $Z_{r_{\max}} = \{y_{\text{top}}\} = Y_{r_{\max}}$, and thus

Procedure Approx-NNS (Input: $q \in X$ and $\epsilon > 0$).
1. set $r = r_{\max}$ and $Z_r = \{y_{\text{top}}\}$.
2. **while** $2r(1 + 1/\epsilon) > d(q, Z_r)$ and Z_r is proper **do**
3. set $Z_{r/2} = \{y \in \bigcup_{z \in Z_r} L_{z,r} : d(q, y) \le d(q, Z_r) + r\}$.
4. set $r = r/2$.
5. return $z \in Z_r$ for which $d(q, z)$ is minimal.

Figure 1: Approximate nearest neighbor procedure.

for a point a that is closest to q among S we indeed have by Lemma 2.1 that $d(a, Z_{r_{\max}}) \le 2r_{\max}$.

For the inductive step, consider the construction of $Z_{r/2}$ in line 3 assuming that Z_r satisfies the induction hypothesis. It follows that Z_r contains a point z with $d(z, a) \le 2r$. By Lemma 2.1, $Y_{r/2}$ contains a point y with $d(a, y) \le r$, so $d(z, y) \le d(z, a) + d(a, y) \le 2r + r = 3r$, and thus $y \in L_{z,r}$ (y appears in the navigation list of z since $\gamma \ge 6$). To complete the proof, it suffices to show that y is included in $Z_{r/2}$. Indeed, observe that when $Z_{r/2}$ is constructed in line 3, $d(q, y) \le d(q, a) + d(a, y) \le d(q, Z_r) + r$, so $Z_{r/2}$ will include y. □

Lemma 2.5. *Procedure* Approx-NNS *outputs a point whose distance from q is at most $(1 + \epsilon)\, d(q, S)$.*

Proof. Let r^* be the value of r at the end of the procedure. It suffices to prove that $d(q, Z_{r^*}) \le (1 + \epsilon)\, d(q, S)$, because then the proof of the lemma follows from the choice made in line 5. There are two conditions in line 2 that may cause the while-loop to stop; consider first the case when $2r^*(1 + 1/\epsilon) \le d(q, Z_{r^*})$. By Lemma 2.4 we know that $d(q, Z_{r^*}) \le d(q, S) + 2r^*$. Combining the last two inequalities we get $2r^*/\epsilon \le d(q, S)$. Plugging this back into the second inequality, we get $d(q, Z_{r^*}) \le d(q, S) + 2r^* \le (1 + \epsilon)\, d(q, S)$, as desired.

Consider next the case when Z_{r^*} is non-proper. We may also assume that $d(q, Z_{r^*}) > 0$, as otherwise we are done. For the sake of analysis, suppose that we were to continue iterating (i.e., ignore the requirement of line 2 that Z_r is proper). Notice that this process cannot go forever, because $d(q, Z_r) \ge d(q, S) > 0$ while $2r(1 + 1/\epsilon) \to 0$. Furthermore, the point that would have been returned by the modified procedure is exactly the one returned by the actual procedure, because the sets Z_r that would have been constructed further (i.e., for $r < r^*$) would all contain the same single point as Z_{r^*}. Finally, it is easy to see that our analysis above for the case $d(q, Z_{r^*}) \le (1 + \epsilon)\, d(q, S)$ (including Lemma 2.4) is applicable also to the modified procedure, and we obtain that in both the modified and actual procedure, $d(q, Z_{r^*}) \le (1 + \epsilon)\, d(q, S)$ as desired. In fact, if the iteration stops with Z_r being non-proper, then the procedure outputs the optimal (unique) closest point to q among S, because we can apply the same "modified procedure" argument with an arbitrarily small $\epsilon > 0$. □

Lemma 2.6. *Procedure* Approx-NNS *runs in time* $2^{O(\dim(S))} \log \Delta + (1/\epsilon)^{O(\dim(S))}$.

Proof. We first bound the size of a set Z_r computed by Procedure Approx-NNS. The set Z_r of line 1 is trivial, so consider a set $Z_{r/2}$ constructed in line 3. For all $y \in Z_{r/2}$, the conditions in lines 2 and 3 imply $d(q, y) \le d(q, Z_r) + r < 2r(1+1/\epsilon) + r = r(3+2/\epsilon)$. Since the distance between any two points of $Z_{r/2} \subseteq Y_{r/2}$ is at least $r/2$, we have by Lemma 1.2 that $|Z_{r/2}| \le (2 + 1/\epsilon)^{O(\dim(S))}$.

A cruder time bound of $(2 + 1/\epsilon)^{O(\dim(S))} \log \Delta$ follows by bounding separately the number of iterations that the procedure performs and the running time of a single iteration. For the first bound (number of iterations), observe that once r becomes smaller than $d_{\min}/(6 + 6/\epsilon)$, the diameter of $Z_{r/2}$ is (by lines 2 and 3) at most $2[d(q, Z_r) + r] < 2[2r(1 + 1/\epsilon) + r] < d_{\min}$, and thus $Z_r \subseteq S$ contains at most one point. If, in addition, $r < d_{\min}/\gamma$, then Z_r must be non-proper, and the second condition in line 2 does not hold. Hence, the scales r iterated over are between $r_{\max} = O(d_{\max})$ and $\Omega(d_{\min}/[\gamma + 1 + 1/\epsilon])$, and thus the number of iterations is at most $\log \Delta + \log(1 + 1/\epsilon) + O(1)$. We now show the second bound (running time of a single iteration). An iteration scans $|Z_r|$ lists $L_{z,r}$ of length $|L_{z,r}| \le 2^{O(\dim(S))}$ each, and computes their union (which requires the removal of duplicates and can be done by sorting). It follows that the running time of a single iteration is at most $O(|Z_r| \cdot |L_{z,r}| \log(|Z_r| \cdot |L_{z,r}|)) \le (2+1/\epsilon)^{O(\dim(S))}$. A similar scan is performed in line 5, with running time $O(|Z_r|)$. Altogether, the total running time of the procedure is indeed at most $(2 + 1/\epsilon)^{O(\dim(S))} \log \Delta$.

Employing a more careful analysis, we shall now improve the upper bound to that stated in the lemma. We may assume $\epsilon < 1$, as otherwise it's the same as the cruder bound. For iterations in which $r \geq d(q,S)$, we can bound the diameter of Z_r (using line 3 and Lemma 2.4) by $2[d(q,Z_{2r})+2r] \leq 2[d(q,S)+4r+2r] \leq 14r$, and using Lemma 1.2 we get that $|Z_r| \leq 2^{O(\dim(S))}$, regardless of ϵ. The number of such iterations is bounded, as above, by $\log \Delta + \log(1/\epsilon) + O(1)$. For iterations in which $r < d(q,S)$, we use the weaker bound from above $|Z_r| \leq (2+1/\epsilon)^{O(\dim(S))}$, However, the number of these iterations is at most $\log(1/\epsilon) + O(1)$, because in every iteration, the condition in line 2 guarantees that $d(q,Z_r) < 4r/\epsilon$, i.e., $r > \epsilon\, d(q,Z_r)/4 \geq \epsilon\, d(q,S)/4$. We conclude that the total running time is at most $2^{O(\dim(S))} \log \Delta + (1/\epsilon)^{O(\dim(S))}$.

Notice that we analyzed above only the number of operations performed by Procedure Approx-NNS explicitly. In general, locating a particular navigation list for a given point requires time $O(\log\log\Delta)$. However, we can reduce this cost as per the remark in Section 2.1. Since all the accesses to navigation lists are via other navigation lists, we can access each list in $O(1)$ by maintaining, for each scale r navigation list, direct pointers to the scale $r/2$ lists. $\square$

2.4 Range queries and exact nearest neighbor. We can use the data structure to implement a range query operation: Given a point $q \in X$ and a distance $t > 0$, this operation returns the set of points $B(q,t)$. Our upper bound on the running time of this operation depends linearly on $|B(q,O(t))|$. In many cases it is plausible that this quantity is not much larger than the output length $|B(q,t)|$, although the ratio between the two can be arbitrarily large in the worst-case, even in doubling metrics. It is then straightforward to combine the range query operation with the 2-NNS algorithm of Section 2.3 to arrive at an exact nearest neighbor search procedure.

Theorem 2.3. *A range query within distance t around a point q can be computed using the data structure in time $2^{O(\dim(S))}(\log \Delta + |B(q,O(t))|)$.*

Proof (Sketch). By rounding, we may assume that $t \in \Gamma$. The range query procedure iterates over the scales $r = r_{\max}, \ldots, r_{\min}$, and constructs a set $Z_r \subseteq Y_r$ for each such value of r. The first phase corresponds to scales $r = r_{\max}, \ldots, 2t$ and is similar to Section 2.3—the set $Z_r \subseteq Y_r$ contains the points of Y_r nearby q. In the second phase, which corresponds to scales $r = t, \ldots, r_{\min}$, the set Z_r contains all the points that can be found by scanning the scale $2r$ navigation lists of all the points in Z_{2r}. The procedure reports all the points that are found in the second phase whose distance from q is at most t.

Let us show that the procedure indeed reports any point $s \in B(q,t)$. By Lemma 2.1, there is a point $y \in Y_t$ such that $d(s,y) \leq 2t$. For the sake of analysis, let a be a closest point to q among those in S. Then $d(q,a) \leq d(q,s) \leq t$. By Lemma 2.4, there is a point $z \in Z_{2t}$ such that $d(a,z) \leq 4t$. Altogether, we have $d(z,y) \leq d(z,a) + d(a,q) + d(q,s) + d(s,y) \leq 4t + t + t + 2t \leq \gamma \cdot 2t$, and hence $y \in Y_t$ must appear in the list $L_{z,2t}$. Now it is easy to see that the second phase of the algorithm, which recursively scans all navigation lists reachable from Z_{2t}, must find s along the way, due to the same argument as in Lemma 2.1.

To analyze the running time, observe that Z_{2t} is contained, because of its construction together with Lemma 2.4, in a ball of radius $d(q,Z_{4t}) + 2t = O(t)$ around q. It follows that any point found in the second phase is within distance $\gamma(2t+t+t/2+\cdots) \leq 4\gamma t$ from Z_{2t}, and hence within distance $O(t)$ from q. We can then use an adaptation of Theorem 2.1 to bound the running time of the second stage by $2^{O(\dim(S))}|B(q,O(t))|$. In order to bound the running time of the first phase we need to modify it so that the iterations are stopped and the empty set is reported, whenever $d(q,Z_r) > 3r$. This implies that the diameter of Z_r is at most $2\,d(q,Z_r)+r = O(r)$, and thus $|Z_r| \leq 2^{O(\dim(S))}$ (by Lemma 1.2). Hence, the first phase runs in time $2^{O(\dim(S))} \log \Delta$. Observe that if there is a point $s \in B(q,t)$, the first phase would not stop before reaching scale $2t$, because by Lemma 2.4, $d(q,Z_r) \leq d(q,s) + 2r \leq 3r$ for every $r \geq 2t$. $\square$

Theorem 2.4. *An exact nearest neighbor search for a query q can be computed using the data structure in time $2^{O(\dim(S))}(\log \Delta + |B(q,O(d(q,S)))|)$.*

Proof. Given a query q, we first apply Theorem 2.2 to compute a 2-NNS for q, i.e., find a point $s_q \in S$ such that $d(q,s_q) \leq 2\,d(q,S)$. Now we apply Theorem 2.3 to find all the points within distance $t = d(q,s_q)$ from q, and report the one or more points among them that are closest to q. It is immediate that this algorithm indeed finds all the closest points to q and that the running time bound is as claimed. $\square$

2.5 Inserting a point. We now show a procedure that updates the data structure when a new point $q \in M$ is inserted to S.

Theorem 2.5. *The data structure can be updated with an insertion of a point to S in time $2^{O(\dim(S))} \log \Delta \log\log \Delta$. This includes $2^{O(\dim(S))} \log \Delta$ distance computations.*

Proof (Sketch). The main idea is that regardless of how the nets were constructed, it is possible to update each

r-net Y_r by either adding q to it or leaving it unchanged. Indeed, this can be seen by induction on r. The base case is a sufficiently small scale r, for which Y_r contains, by definition, all the points, and thus q must be added to this net. For the inductive step, assume first that the update to $Y_{r/2}$ leaves it unchanged; then it is clear that Y_r need not be changed. Assume next that a net $Y_{r/2}$ is updated by adding q to it; then we update Y_r by adding q to Y_r if and only if $d(q, Y_r) \geq r$. This guarantees that Y_r remains an r-net of $Y_{r/2}$ (although there may be other ways to update Y_r). Deciding whether $d(q, Y_r) \geq r$ will be done by using a set Z_r that contains (similar to Section 2.3) all points of Y_r that are nearby q. Recall that the net Y_r is only maintained through its navigation lists, so adding q to Y_r actually requires us to construct a scale r navigation list for q and to update the scale $2r$ lists of nearby points. Again, this task is carried out using the aforementioned sets Z_r. $\square$

2.6 Deleting a point. Our data structure can also support deletion of points. The procedure for deleting a point is generally similar to that of inserting a point (Section 2.5). The main technical difference occurs when a point q is deleted from a net Y_r. In this case, $Y_r \setminus \{q\}$ need not not be an r-net of $Y_{r/2} \setminus \{q\}$, in which case it is necessary to promote (i.e., add) to Y_r one or more points of $Y_{r/2}$. However, this decision (and the corresponding updates to various lists) can be done using the relatively small sets Z_r that contain all the points of Y_r that are nearby q. Details are omitted from this version of the paper.

2.7 Improved bounds for KR metrics. Clearly, by Lemma 1.1, one may replace $\dim(S)$ by $\dim_{\mathrm{KR}}(S)$ in all the aforementioned complexity bounds. We now show that, using essentially the same data structure, we can achieve improved performance under the additional assumption that $\dim_{\mathrm{KR}}(S \cup \{q\})$ is small, where q is the query point. First, in the time complexity of the various operations, we are able to replace Δ with $n = |S|$. Secondly, we show that an exact nearest neighbor can be found (rather than only a $(1+\epsilon)$-approximation). In fact, the k closest points can be found in time $O(\log n + k)$. These bounds match the guarantees of Karger and Ruhl [KR02], but our data structure is deterministic, dimension oblivious, and requires smaller space. (And, of course, extends to metrics with bounded doubling dimension.)

Theorem 2.6. *The $(1+\epsilon)$-approximate nearest neighbor algorithm from Theorem 2.2 runs in time $2^{O(\dim_{\mathrm{KR}}(S\cup\{q\}))} \log n + (1/\epsilon)^{O(\dim_{\mathrm{KR}}(S\cup\{q\}))}$. Similarly, the insertion and deletion procedures of Theorem 2.5 run in time $2^{O(\dim_{\mathrm{KR}}(S\cup\{q\}))} \log n \log\log n$.*

The proof of Theorem 2.6 follows from Lemma 2.8 below; details are omitted from this version.

Lemma 2.7. *Let x, y be points in a metric (X, d) and set $r = \frac{1}{3} d(x, y)$; then $|B(x, r)| \geq 2^{-2\dim_{\mathrm{KR}}(X)} |B(y, r)|$.*

Proof. Observe that $B(y, r) \subseteq B(x, 3r + r)$ and thus $|B(y, r)| \leq |B(x, 4r)| \leq 2^{2\dim_{\mathrm{KR}}(X)} |B(x, r)|$. $\square$

Lemma 2.8. *Every point $x \in S$ has at most $2^{O(\dim_{\mathrm{KR}}(S))} \log n$ non-trivial navigation lists $L_{x,r}$.*

Proof. Suppose that x has non-trivial navigation lists for scales $r_0 < r_1 < \ldots < r_t$. By definition, each such navigation list L_{x,r_i} contains a point $y_i \in Y_{r_i/2}$ that is different from x, and with $d(x, y_i) \leq \gamma r_i$. By definition, $x \in Y_{r_i} \subseteq Y_{r_i/2}$ and thus $d(x, y_i) \geq r_i/2$.

We now claim that if $c = c(\gamma)$ is a suitable constant, then $|B(x, r_{i+c})| \geq (1 + 2^{-2\dim_{\mathrm{KR}}(X)}) |B(x, r_i)|$ for every $i \geq 0$. Given this claim, the lemma follows easily; by induction, $|B(x, r_{4i})| \geq (1 + 2^{-2\dim_{\mathrm{KR}}(X)})^i$ (the base case $i = 0$ is obvious), and since $|S| = n$ we get that the number of distinct scales r_i is $t + 1 \leq 2^{O(\dim_{\mathrm{KR}}(X)} \log n$.

It remains to prove the above claim. By applying Lemma 2.7 for the points x and y_{i+3} with radius $r = d(x, y_{i+3})/3$, we get that $|B(y_{i+3}, r)| \geq 2^{-2\dim_{\mathrm{KR}}(X)} |B(x, r)|$. By definition, these two balls are disjoint, and they are both contained in a radius $4r$ ball centered at x. Thus, $|B(x, 4r)| \geq |B(y_{i+3}, r)| + |B(x, r)| \geq (1 + 2^{-2\dim_{\mathrm{KR}}(X)}) |B(x, r)|$. The claimed inequality now follows by plugging in the upper bound $4r = \frac{4}{3} d(x, y_{i+3}) \leq \frac{4}{3} \gamma r_{i+3} \leq r_{i+c}$ (assuming $2^{c-3} \geq \frac{4}{3}\gamma$) and the lower bound $r = d(x, y_{i+3})/3 \geq r_{i+3}/6 > r_i$. $\square$

Theorem 2.7. *The k nearest neighbors of a query q can be computed using the data structure in time $2^{O(\dim_{\mathrm{KR}}(S\cup\{q\}))}(k + \log n)$. In particular, exact* NNS *can be computed in time $O(\log n)$ whenver $\dim_{\mathrm{KR}}(S \cup \{q\}) = O(1)$.*

The proof of this theorem is based on applying Theorem 2.3, and bounding the running time using arguments like in Lemma 2.8. Details aomitted from this version of the paper.

3 Lower bounds

This section shows that the complexity of our data structure is nearly optimal, by lower bounding the number of distance computations that are necessary (information theoretically) to answer NNS and $(1+\epsilon)$-NNS queries. The two lower bounds presented below assume that computing the distance between two points is the only costly operation, and that no information about the distances can be deduced by other means, such as hashing points' identifiers.[2]

Formally, suppose that the distance between every two points in S is known, and that the distance between the query point q and any point in S requires an access to an oracle. We consider an adversarial oracle, i.e., we examine the worst-case complexity of answering a $(1+\epsilon)$-NNS query (over all possible oracles). For simplicity, we state the lower bound for $\epsilon \leq 2$, although the proof immediately extends to larger ϵ. These results hold even for randomized algorithms (both Las-Vegas and Monte-Carlo).

LEMMA 3.1. *There is an input data set S for the distance oracle model, such that even though S is doubling and $\dim_{\mathrm{KR}}(S) = O(1)$, any exact* NNS *algorithm must access the oracle $\Omega(n)$ times (for a worst-case query).*

Proof (Sketch). Let (S, d) be the metric on the integer points $1, \ldots, n$ on the real line. Clearly, $\dim(S) = O(1)$. Let the query point q be at distance $n-1$ from a single point $i \in S$ and at distance n from all the other points of S. Obviously, any deterministic NNS algorithm must report the point i. It follows that if the value of i is chosen adversarially, then the NNS algorithm must compute n distances, in the worst-case, in order to find i. The proof for randomized algorithms is similar, using Yao's minimax principle. $\square$

LEMMA 3.2. *There is an input data set S such that any* $(1+\epsilon)$-NNS *algorithm (for a fixed $0 \leq \epsilon \leq 1$) that works in the distance oracle model, must access the oracle at least $2^{\Omega(\dim(S))} \log n$ times (for a worst-case query).*

We omit the proof from this version. It is based on a data set S which is the shortest path metric between the leaves of a complete λ-ary tree metric, in which the length of edges at depth i is $1/2^i$. It shows that any algorithm essentially has to perform a linear search among the children of the root (the vertex at depth 0), then a linear search among the children of a vertex at depth 1, and so forth.

References

[AMN+98] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[Ass83] P. Assouad. Plongements lipschitziens dans $\mathbf{R}^n$. *Bull. Soc. Math. France*, 111(4):429–448, 1983.

[Bri95] S. Brin. Near neighbor search in large metric spaces. In *21st International Conference on Very Large Data Bases*, pages 574–584, 1995.

[Cla99] K. L. Clarkson. Nearest neighbor queries in metric spaces. *Discrete Comput. Geom.*, 22(1):63–93, 1999.

[CNBYM01] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Proximity searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, September 2001.

[FK97] C. Faloutsos and I. Kamel. Relaxing the uniformity and independence assumptions using the concept of fractal dimension. *J. Comput. System Sci.*, 55(2):229–240, 1997.

[GKL03] A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low–distortion embeddings. Accepted to 43rd Symposium on Foundations of Computer Science, 2003.

[Hei01] J. Heinonen. *Lectures on analysis on metric spaces*. Universitext. Springer-Verlag, New York, 2001.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *30th Annual ACM Symposium on Theory of Computing*, pages 604–613, May 1998.

[KOR98] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high-dimensional spaces. In *30th Annual ACM Symposium on Theory of Computing*, pages 614–623. ACM, 1998.

[KR02] D. Karger and M. Ruhl. Finding nearest neighbors in growth-restricted metrics. In *34th Annual ACM Symposium on the Theory of Computing*, pages 63–66, 2002.

A Appendix

Proof. [Lemma 1.1] Let K be the KR-constant of X and fix some ball $B(x, 2r)$. We will show that $B(x, 2r)$ can be covered by K^4 balls of radius r. It will follow that $\dim(X) \leq 4 \log_2 K = 4 \cdot \dim_{\mathrm{KR}}(X)$.

Let Y be an r-net of $B(x, 2r)$, then

$$B(x, 2r) \subset \bigcup_{y \in Y} B(y, r) \subset B(x, 4r).$$

Also, for every $y \in Y$, $|B(x, 4r)| \leq |B(y, 8r)| \leq K^4 \, |B(y, \frac{r}{2})|$. Since $B(y, \frac{r}{2})$ and $B(y', \frac{r}{2})$ are disjoint for $y \neq y' \in Y$, it follows that $|Y| \leq K^4$. We conclude that the K^4 balls $\{B(y, r) : y \in Y\}$ cover $B(x, 2r)$.

[2]This is similar in spirit to lower bounds on the number of comparisons in sorting.