Finding Friends and Followers in Sub-linear Time

David Arthur, Steve Y. Oudot, Anneesh Sharma

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Finding Friends and Followers in Sub-linear Time

David Arthur — Steve Y. Oudot — Aneesh Sharma

N° 7084

Novembre 2009

ISSN 0249-6399 ISRN INRIA/RR--7084--FR+ENG

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
INRIA
centre de recherche SACLAY - ÎLE-DE-FRANCE

Finding Friends and Followers in Sub-linear Time

David Arthur* , Steve Y. Oudot†, Aneesh Sharma‡

Thème : SYM — Systèmes symboliques
Équipe-Projet Geometrica

Rapport de recherche n° 7084 — Novembre 2009 — 23 pages

Abstract: The approximate Nearest Neighbor ($\mathcal{NN}$) search problem asks to pre-process a given set of points P in such a way that, given any query point q, one can retrieve a point in P that is *approximately* closest to q. Of particular interest is the case of points lying in high dimensions, which has seen rapid developments since the introduction of the Locality-Sensitive Hashing (LSH) data structure by Indyk and Motwani [20]. Combined with a space decomposition by Har-Peled [17], the LSH data structure can answer approximate $\mathcal{NN}$ queries in sub-linear time using polynomial (in both d and n) space. Unfortunately, it is not known whether Har-Peled's space decomposition can be maintained efficiently under point insertions and deletions, so the above solution only works in a static setting. In this paper we present a variant of Har-Peled's decomposition, based on random semi-regular grids, which can achieve the same query time with the added advantage that it can be maintained efficiently even under adversarial point insertions and deletions. The outcome is a new data structure to answer approximate $\mathcal{NN}$ queries efficiently in dynamic settings.

Another related problem known as Reverse Nearest Neighbor ($\mathcal{RNN}$) search is to find the *influence set* of a given query point q, i.e. the subset of points of P that have q as their nearest neighbor. Although this problem finds many practical applications, very little is known about its complexity. In particular, no algorithm is known to solve it in high dimensions in sub-linear time using sub-exponential space. In this paper we show how to pre-process the data points, so that Har-Peled's space decomposition combined with modified LSH data structures can solve an approximate variant of the $\mathcal{RNN}$ problem efficiently, using polynomial space. The query time of our approach is bounded by two terms: the first one is sub-linear in the size of P and corresponds roughly to the incompressible time needed to locate the query point in the data structure; the second one is proportional to the size of the output, which is a set of points as opposed to a single point for (approximate) $\mathcal{NN}$ queries. An interesting feature of our $\mathcal{RNN}$ solution is that it is flexible enough to be applied indifferently in monochromatic or bichromatic settings.

Key-words: nearest neighbor search, reverse nearest neighbor search, locality-sensitive hashing.

* Department of Computer Science, Stanford University. Email: darthur@cs.stanford.edu
† Geometrica group, INRIA Saclay – Île-de-France. Email: steve.oudot@inria.fr
‡ Institute for Computational and Mathematical Engineering, Stanford University. Email: aneeshs@cs.stanford.edu

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université
4, rue Jacques Monod, 91893 ORSAY Cedex
Téléphone : +33 1 72 92 59 00

Trouver ses amis et ses partisans en temps sous-linéaire

Résumé : Le problème de la recherche de plus proches voisins ($\mathcal{NN}$) consiste à préparer un nuage de points P donné de manière à pouvoir, pour n'importe quel point de requête q, retrouver un point de P qui est *approximativement* le plus proche de q. Le cas des nuages de points en grande dimension présente un intérêt particulier, et son analyse s'est rapidement développée à la suite de l'introduction du *Locality-Sensitive Hashing* (LSH) par Indyk et Motwani [20]. Combinée avec la décomposition spatiale proposée par Har-Peled [17], cette technique permet de répondre à des requêtes de plus proches voisins en temps sous-linéaire avec une consommation en mémoire qui reste polynômiale à la fois en d et en la taille n du nuage. Malheureusement, on ne sait toujours pas si la décomposition spatiale d'Har-Peled peut être maintenue efficacement après insertion ou suppression de points dans le nuage, ce qui fait que la solution au problème $\mathcal{NN}$ mentionnée ci-dessus ne marche que dans un cas statique. Dans cet article nous proposons une variante de la décomposition d'Har-Peled, fondée sur l'utilisation de grilles semi-régulières aléatoires, qui permet d'obtenir les mêmes temps de requête tout en étant efficacement maintenue après insertion ou suppression dans le nuage.

Un autre problème lié est la recherche de plus proches voisins inverses ($\mathcal{RNN}$), dont le but est de trouver la *zone d'influence* d'un point de requête q donné, c'est-à-dire l'ensemble des points de P qui ont q pour plus proche voisin dans le nuage $P \cup \{q\}$. Alors même que ce problème trouve de nombreuses applications pratiques, sa complexité théorique est très peu connue. En particulier, il n'existe pas à l'heure actuelle d'algorithme le résolvant à la fois en temps sous-linéaire et en espace sous-exponentiel. Dans cet article nous montrons comment préparer le nuage de points P de manière à ce que la décomposition spatiale d'Har-Peled, combinée à des structures LSH modifiées, puisse résoudre efficacement une version approchée du problème $\mathcal{RNN}$, en utilisant une quantitée de mémoire polynômiale. Le temps de requête de notre approche est borné par deux termes : le premier correspond en gros au temps incompressible nécessaire à la localisation du point de requête dans la structure de donnée ; le deuxième est proportionnel à la taille de la réponse, qui est un ensemble de points et non pas un unique point comme dans le cas de $\mathcal{NN}$. Un aspect intéressant de notre approche est qu'elle est suffisamment flexible pour pouvoir être utilisée indifféremment dans un contexte statique ou dynamique.

Mots-clés : recherche de plus proche voisin, recherche de plus proches voisins inverses, hachage géométrique

1 Introduction

Proximity queries are ubiquitous in science and engineering, and given their natural importance they have received a lot of attention from the computer science community [10, 12, 19, 29]. From a theoretical point of view, the most studied of them is the Nearest Neighbor ($\mathcal{NN}$) query, defined as follows: given a metric space (X, d), a point set $P \subseteq X$ and a query point $q \in X$, find the point of $P \setminus \{q\}$ that is closest to q. This point is called the *nearest neighbor* of q with respect to P and is denoted by $\mathcal{NN}_P(q)$. The $\mathcal{NN}$ query can be answered in linear time[1] by brute force search, so the algorithmic challenge is to pre-process the data points so as to be able to answer nearest neighbor queries in sub-linear time. This emphasis on reducing the query time stems from the fact that typical applications require to answer many $\mathcal{NN}$ queries on the same point set (see Shakhnarovich et al. [29] for a list of such applications). Although several efficient algorithms are known for the $\mathcal{NN}$ problem when the dimension is low [6], there is no known algorithm that can answer the $\mathcal{NN}$ query in sub-linear time using sub-exponential space (in both the size n of the point set P and dimension d of the metric space X). This conundrum is usually referred to as the *curse of dimensionality*.

In light of the apparent hardness of the $\mathcal{NN}$ problem, researchers have proposed an approximate version called ε-$\mathcal{NN}$, where the answer can be any point of $P \setminus \{q\}$ whose distance to the query point is within a factor $(1 + \varepsilon)$ of the distance from q to its true nearest neighbor [4, 11, 20, 22, 25]. This version of the problem indeed proved to be more amenable to a solution, and inspired from the random projection techniques developed by Kleinberg [22], two breakthrough results for the ε-$\mathcal{NN}$ problem were obtained independently and on the same year by Kushilevitz et al. [25] and by Indyk and Motwani [20]. Both of these papers gave data structures that can answer ε-$\mathcal{NN}$ queries with truly sublinear (in n) runtime and use space that is polynomial in n, d and $1/\varepsilon$. The currently known fastest algorithms for ε-$\mathcal{NN}$ search in high dimensions (see Andoni and Indyk [3] for a survey treatment) are based on the idea of Locality-Sensitive Hashing (LSH), first introduced by Indyk and Motwani [20]. In their seminal paper, they reduced the ε-$\mathcal{NN}$ problem to a decision version that involves an additional input parameter r and (informally speaking) asks to decide whether the distance from the query point q to the point set P is at most r, greater than $(1 + \varepsilon)r$, or somewhere in-between. To be more specific, their space decomposition reduces the ε-$\mathcal{NN}$ query to a poly-logarithmic number of (r, ε)-$\mathcal{NN}$ queries, for some suitable choices of r. Moreover, they showed how to use LSH to answer each (r, ε)-query in time[2] $\tilde{O}(n^{\rho})$ for some constant $\rho < 1$, thus providing a fully sublinear-time procedure for solving ε-$\mathcal{NN}$ queries. The LSH technique was originally designed for the Hamming cube, but later work by Datar et al. [13] and Andoni and Indyk [2] provided locality-sensitive hash functions for affine spaces $\mathbb{R}^d$ equipped with ℓ_p-norms, $p \in [0, 2]$.

The space decomposition of Indyk and Motwani [20] is rather involved, and in fact Har-Peled [17] gave a much simpler space decomposition that works in any metric space and enables the same kind of reduction from ε-$\mathcal{NN}$ to a poly-logarithmic number of (r, ε)-$\mathcal{NN}$ queries. Har-Peled's construction consists of a hierarchical tree, each node v of which represents some subset of the data points and contains a logarithmic number of LSH data structures that allow it to efficiently decide whether a given ε-$\mathcal{NN}$ query can be solved at node v; if not, it identifies a child of v in which the algorithm then recurses (see Section 2.2 for details). Since its introduction, Har-Peled's space decomposition has remained virtually untouched, save for a few slight improvements such as the one suggested by Sabharwal et al. [28]. In particular, the question of whether the

[1] Following previous literature [19], we use the number of distance function evaluations as our measure of complexity. In other words, we omit the additional factor (always a polynomial of small degree) induced by the dimension of the space in our bounds.

[2] The big-$\tilde{O}$ notation hides some additional factors that are linear in d and poly-logarithmic in n/ε.

tree structure can be efficiently updated in dynamic settings where points can be added to or deleted from P has remained open until now.

The first contribution of our paper is to provide a positive answer to this question, or rather, to show that a variant of Har-Peled's construction can be made dynamic. While we keep the idea of using a tree structure, we change the way the subset P_v associated with a node v of the tree is partitioned among the children of v. Specifically, instead of using the connected components of some union of balls to form the clusters, we use the cells of some random semi-regular grid. Although less general (the use of grids requires the ambient space to be affine), this new clustering strategy enables us to update the tree structure efficiently under point insertions or deletions. The outcome is a data structure that can answer ε-$\mathcal{NN}$ queries in sub-linear time (in fact with the same query time as in the static setting), with sub-linear amortized insertion and deletion costs.

A popular variant of the $\mathcal{NN}$ query is the reverse nearest neighbor query ($\mathcal{RNN}$), which focuses on the *influence set* of a given query point. More precisely, given a metric space (X, d), a point set $P \subseteq X$ and a query point $q \in X$, the goal is to find the set of the points of $P \setminus \{q\}$ that are closer to q than to any other point of P. This set is called the *reverse nearest neighbor set* of q with respect to P and is denoted by $\mathcal{RNN}_P(q)$. Similar to the $\mathcal{NN}$ query, an approximate version of the $\mathcal{RNN}$ query asks for the set $\mathcal{RNN}_P(q, \varepsilon)$ of the points $p \in P \setminus \{q\}$ such that $\mathrm{d}(p, q) \leq (1 + \varepsilon)\mathrm{d}(p, p')$ for all $p' \in P \setminus \{p\}$. The ε-approximate $\mathcal{RNN}$ query asks to retrieve some set S such that $\mathcal{RNN}_P(q) \subseteq S \subseteq \mathcal{RNN}_P(q, \varepsilon)$. In other words, all the true reverse nearest neighbors of q must be found, plus possibly a controlled number of false positives. Such queries arise in many different contexts, among which the multi-scale reconstruction of high-dimensional point cloud data using witness complexes [7, 16] initially motivated this work. We refer the reader to Korn and Muthukrishnan [23] for a list of other relevant applications.

Although ε-$\mathcal{RNN}$ has received a lot of attention from applied scientific communities, who devised numerous heuristics that behave well on practical data (see e.g. [9, 1, 5, 14, 21, 23, 24, 30, 31, 32, 33]), very little is known about the theoretical complexity of the problem. To date, provably efficient methods exist only for low-dimensional settings [8, 26]. Part of the problem is that, in contrast with ε-$\mathcal{NN}$, the answer of the ε-$\mathcal{RNN}$ query is not a single point but a set of points, whose size can be exponential in the dimension [27], so there is no way to achieve a systematic sub-linear query time. Ideally, one would like to achieve a query time of the form[3] $\tilde{O}(n^\rho + |\mathcal{RNN}_P(q)|)$, where ρ is a constant less than 1 and $|\mathcal{RNN}_P(q)|$ is the size of the true reverse nearest neighbors set. Intuitively, the first term represents the incompressible time needed to locate the query point q with respect to P, as in a standard $\mathcal{NN}$ query, while the second term represents the size of the sought-for answer. The second contribution of our paper is to show that by bucketing the data points according to their nearest neighbor distance in P, and by building a modified LSH data structure on each bucket, one can reduce static ε-$\mathcal{RNN}$ queries to a small number of ε-$\mathcal{NN}$ and (r, ε)-$\mathcal{NN}$ queries, and thereby achieve a query time of $\tilde{O}(\frac{1}{\varepsilon} n^\rho + |\mathcal{RNN}_P(q, O(\varepsilon))|)$. Here, the second term comes from the fact that our approach outputs a superset of $\mathcal{RNN}_P(q)$, whose size remains controlled. An interesting feature of our approach is to use Har-Peled's tree structure as a black box, which makes it possible to replace it by our own in dynamic settings, in order to handle point insertions and deletions more efficiently.

We note that the above two problems are embedded in the general setting of maintaining a directed graph $G(P, E)$ where the vertices of the graph are the points of a given point set P, and there is an edge from $p \in P$ to $p' \in P$ if and only if p' is an approximate nearest neighbor of p (in this case we say that p' is a *friend* of p and p is a *follower* of p'). Observe that these relationships are asymmetric.

[3] Here again, the big-$\tilde{O}$ may hide some additional factors that are linear in d and poly-logarithmic in n/ε.

Our contributions. In summary, given a set P of n points in a metric space (X, d), we provide:

- A fully polynomial-size data structure to answer static ε-$\mathcal{RNN}$ queries in $\tilde{O}(\frac{1}{\varepsilon}n^{\frac{1}{1+\varepsilon}} + |\mathcal{RNN}_P(q, O(\varepsilon))|)$ time (Section 3),
- When $(X, \mathrm{d}) = (\mathbb{R}^d, \ell_2)$, a variant of Har-Peled's tree structure that supports insertions and deletions in amortized $\tilde{O}(\frac{1}{\varepsilon}n^{\frac{1}{1+\varepsilon}})$ time (Section 4). As a result, we obtain a fully polynomial-size data structure to answer ε-$\mathcal{NN}$ queries in $\tilde{O}(\frac{1}{\varepsilon}n^{\frac{1}{1+\varepsilon}})$ time, which supports point insertions and deletions in amortized sub-linear time (Section 5).

Our ε-$\mathcal{RNN}$ data structure can also be used in dynamic settings, where it currently supports insertions and deletions in amortized linear time, allowing us to maintain the directed graph mentioned above. Ideally we would like to update the graph in sub-linear time, but this question is currently open (see Section 5). We do note that our ε-$\mathcal{RNN}$ data structures work in the more general bichromatic setting where the sets of potential *friends* and potential *followers* are different. Thus, we follow the bichromatic setting in Section 3.

2 Preliminaries

In Section 2.1 we introduce some useful notation and state the approximate nearest neighbor and reverse nearest neighbor problems formally. In Sections 2.2 through 2.4 we give an overview of LSH and its application to approximate nearest neighbor search. The various data structures and algorithms introduced in this section will be used as black boxes in Section 3.

2.1 Problem statements and notations

Throughout the paper, (X, d) denotes a metric space and P a finite subset of X. Given a point $x \in X$ and a real parameter $\varepsilon \geq 0$, let $d_{\mathcal{NN}_P}(x, \varepsilon)$ be equal to $(1+\varepsilon)$ times the distance of x to $P \setminus \{x\}$, that is: $d_{\mathcal{NN}_P}(x, \varepsilon) = (1+\varepsilon)\min\{\mathrm{d}(x, p) \mid p \in P \setminus \{x\}\}$. Let also $\mathcal{NN}_P(x, \varepsilon)$ denote the set of points of $P \setminus \{x\}$ that lie within distance $d_{\mathcal{NN}_P}(x, \varepsilon)$ of x. Then, $d_{\mathcal{NN}_P}(x, 0)$ is simply the distance of x to $P \setminus \{x\}$, while $\mathcal{NN}_P(x, 0)$ is the set of nearest neighbors of x in $P \setminus \{x\}$. We denote them by $d_{\mathcal{NN}_P}(x)$ and $\mathcal{NN}_P(x)$ respectively in the sequel, for simplicity.

Problem 1 (ε-$\mathcal{NN}$)**.** *Given a query point $q \in X$, the* ε-nearest neighbor query *asks to retrieve any point $p \in \mathcal{NN}_P(q, \varepsilon)$.*

Consider now two finite subsets R, B of X. These will be referred to as the red and blue sets in the sequel. Given a point $x \in X$ and a real parameter $\varepsilon \geq 0$, we denote by $\mathcal{RNN}_{B,R}(x, \varepsilon)$ the set of blue points that have x among their ε-approximate neighbors in the augmented red set $R \cup \{x\}$, that is: $\mathcal{RNN}_{B,R}(x, \varepsilon) = \left\{p \in B \setminus \{x\} \;\middle|\; x \in \mathcal{NN}_{R\cup\{x\}}(p, \varepsilon)\right\}$. Equivalently, $\mathcal{RNN}_{B,R}(x, \varepsilon)$ is the set of all points $p \in B$ such that $\mathrm{d}(p, x) \leq d_{\mathcal{NN}_R}(p, \varepsilon)$. When $\varepsilon = 0$, it is the set of reverse nearest neighbors of x in this bichromatic setting, noted $\mathcal{RNN}_{B,R}(x)$ for simplicity.

Problem 2 (ε-$\mathcal{RNN}$)**.** *Given a query point $q \in X$, the* ε-approximate reverse nearest neighbor query *asks to return any set $S \subseteq B \setminus \{q\}$ satisfying the following nesting property: $\mathcal{RNN}_{B,R}(q) \subseteq S \subseteq \mathcal{RNN}_{B,R}(q, \varepsilon)$.*

Intuitively, the goal is to retrieve all the true reverse nearest neighbors of q, plus possibly a few additional false positives that are *close to being* reverse nearest neighbors of q. Parameter ε controls both the number and the quality of the false positives. When $\varepsilon = 0$, the true reverse nearest neighbor set must be returned.

2.2 Reduction from ε-$\mathcal{NN}$ to the (r, ε)-$\mathcal{NN}$problem

Problem 3 ((r, ε)-$\mathcal{NN}$)**.** *Given a finite set $P \subseteq X$ and a query point $q \in X$, the* (r, ε)-neighbor query *asks the following:*

- *if* $d_{\mathcal{NN}_P}(q) \leq r$*, then return* YES *and any point* $p \in P$ *such that* $\mathrm{d}(p,q) \leq r(1+\varepsilon)$*;*
- *if* $d_{\mathcal{NN}_P}(q) > r(1+\varepsilon)$*, then return* NO*;*
- *else* $(r < d_{\mathcal{NN}_P}(q) \leq r(1+\varepsilon))$*, return any of the above answers.*

This query amounts to deciding approximately whether q lies in the union of balls of same radius r about the points of P. It is therefore known as *ε-approximate Point Location among Equal Balls* (ε-PLEB) in the literature. The original LSH paper [20] showed a construction that reduces the ε-$\mathcal{NN}$ problem to a logarithmic number of (r,ε)-$\mathcal{NN}$ queries. Har-Peled [17] proposed a simpler and more efficient construction based on a divide-and-conquer strategy: it consists in building a tree $\mathcal{T}(P,\varepsilon)$ such that each node v is assigned a subset $P_v \subseteq P$ and an interval $[r_v, R_v]$ of possible values of parameter r. Each ε-$\mathcal{NN}$ query is performed by traversing down the search tree $\mathcal{T}(P,\varepsilon)$, and by answering two (r,ε)-queries at each node v to decide (approximately) whether $d_{\mathcal{NN}_P}(q)$ belongs to the interval $[r_v, R_v]$ or not: in the former case, $O(\log_{1+\varepsilon} \frac{R_v}{r_v})$ queries of type (r,ε)-$\mathcal{NN}$ are sufficient for detecting where $d_{\mathcal{NN}_P}(q)$ lies in the interval and for returning a point of $\mathcal{NN}_P(q,\varepsilon)$; in the latter case, the choice of the child of v in which to continue the recursion is determined by the output of the two (r,ε)-queries. In his construction, Har-Peled ensures that $\frac{R_v}{r_v}$ is at most a polynomial in $\frac{n}{\varepsilon}$ of bounded degree, so that $\log_{1+\varepsilon} \frac{R_v}{r_v} = O(\frac{1}{\varepsilon} \log \frac{n}{\varepsilon})$. More details are provided in Section 4, where we make the data structure dynamic. Note that more involved constructions with reduced space complexities exist [28], but we will stick to Har-Peled's simple construction in the sequel for ease of exposition.

Theorem 2.1 (see Chapter 14 of [17])**.** *Given a finite set $P \subseteq X$ with n points, the tree $\mathcal{T}(P,\varepsilon)$ stores $O(\frac{1}{\varepsilon} \log \frac{n}{\varepsilon})$ data structures for solving (r,ε)-$\mathcal{NN}$ queries per node, and it reduces every ε-$\mathcal{NN}$ query to a set of $O(\frac{1}{\varepsilon} \log \frac{n}{\varepsilon})$ (r,ε)-$\mathcal{NN}$ queries.*

2.3 Solving (r,ε)-$\mathcal{NN}$queries by means of Locality-Sensitive Hashing

Definition 2.2. *Given a metric space (X,d) and a finite set U of integers, given also four parameters $r_1 < r_2$ and $p_1 > p_2$, a family $\mathcal{F} = \{f : X \to U\}$ of hash functions is called* (r_1, r_2, p_1, p_2)-sensitive *if* $\forall x, y \in X$*,*

- $\mathrm{d}(x,y) \leq r_1 \Rightarrow \mathrm{Prob}\left[f(x) = f(y)\right] \geq p_1$*,*
- $\mathrm{d}(x,y) \geq r_2 \Rightarrow \mathrm{Prob}\left[f(x) = f(y)\right] \leq p_2$*,*

where probabilities are given for a random choice of hash function $f \in \mathcal{F}$.

Intuitively, an (r_1, r_2, p_1, p_2)-sensitive family of hash functions can distinguish points that are close together from points that are far away from each other. Such families exist at least when (X,d) is the Hamming cube $\{0,1\}^d$ endowed with the standard Hamming distance [20], or $\mathbb{R}^d$ endowed with an ℓ_p norm such that $p \in [1,2]$ [13].

Provided that a locality-sensitive family of hash functions is given, it is possible to answer (r,ε)-$\mathcal{NN}$ queries (as defined in Problem 3) in sub-linear time [15, 20]. The algorithm proceeds in two phases: in the pre-computation phase, it builds a set of hash tables by hashing the point set using the locality-sensitive family of hash functions, keeping only a single point per non-empty bucket of the hash table; then, in the online query phase, the algorithm hashes the query point q in the hash tables and returns any of the data points that collide with q, if such collisions exist. Otherwise, it answers that q has no (r,ε)-neighbor in the input point set P. This procedure has sub-linear running time and is correct with high probability:

Theorem 2.3 (see [15, 20])**.** *Given a finite set $P \subseteq X$ with n points, two parameters $r, \varepsilon \geq 0$, a $(r,(1+\varepsilon)r,p_1,p_2)$-sensitive family $\mathcal{F} = \{f : P \to U\}$ of hash functions for constants $p_1 > p_2$, and a query point q, it is possible to construct a LSH data structure that solves the (r,ε)-$\mathcal{NN}$ problem with high probability in $O(n^{\frac{1}{1+\varepsilon}} \log n)$ time using $O(n + n^{1+\frac{1}{1+\varepsilon}} \log n)$ space.*

Since each node v of the tree $\mathcal{T}(P,\varepsilon)$ stores $O(\frac{1}{\varepsilon}\log\frac{n}{\varepsilon})$ LSH data structures, each of size $O(|P_v|^{1+\frac{1}{1+\varepsilon}}\log n)$, an easy recursion gives the following bound on the size of $T(P,\varepsilon)$ and associated LSH data structures:

Corollary 2.4. *Combined with the appropriate LSH data structures, $\mathcal{T}(P,\varepsilon)$ can solve ε-$\mathcal{NN}$ queries with high probability in $O\left(\frac{1}{\varepsilon}n^{\frac{1}{1+\varepsilon}}\log n\log\frac{n}{\varepsilon}\right)$ time using $O\left(\frac{1}{\varepsilon}n^{1+\frac{1}{1+\varepsilon}}\mathrm{polylog}\frac{n}{\varepsilon}\right)$ space.*

2.4 The all-(r,ε)-$\mathcal{NN}$ problem

Problem 4 (all-(r,ε)-$\mathcal{NN}$)**.** *Given a finite set $P\subseteq X$ and a query point $q\in X$, the* all-(r,ε)-$\mathcal{NN}$ query *asks to return a set S such that $\mathcal{NN}_P(q,r,0)\subseteq S\subseteq\mathcal{NN}_P(q,r,\varepsilon)$, where by definition $\mathcal{NN}_P(q,r,\eta)=\{p\in P\setminus\{q\}\mid \mathrm{d}(p,q)\leq r(1+\eta)\}$.*

This problem can be seen as a set-theoretic variant of Problem 3, which is more appropriate in the context of reverse nearest neighbor search. When $d_{\mathcal{NN}_P}(q)\leq r$, the output must not be a single point of $\mathcal{NN}_P(q,r,\varepsilon)$ but a whole set of these points that contains at least $\mathcal{NN}_P(q,r,0)$; when $d_{\mathcal{NN}_P}(q)>r(1+\varepsilon)$, $\mathcal{NN}_P(q,r,\varepsilon)$ is empty and so the output must be the empty set, which can be viewed as the set-theoretic equivalent of the NO answer; when $r<d_{\mathcal{NN}_P}(q)\leq r(1+\varepsilon)$, $\mathcal{NN}_P(q,r,0)$ is empty but not $\mathcal{NN}_P(q,r,\varepsilon)$, so any subset of the latter (including the empty set) can be output.

Problem 4 can be solved using a variant of the LSH data structure called $\mathcal{A}(r,\varepsilon)$, which we detail in Appendix A for completeness.

Theorem 2.5. *Given a finite set $P\subseteq X$ with n points, two parameters $r,\varepsilon\geq 0$, an integer $c\geq 2$ and a $(r,(1+\varepsilon)r,p_1,p_2)$-sensitive hash family $\mathcal{F}$ such that $\frac{\ln 1/p_1}{\ln 1/p_2}\leq\frac{1}{1+\varepsilon}$, the $\mathcal{A}(r,\varepsilon)$ data structure answers all-(r,ε)-$\mathcal{NN}$ queries with high probability in $O\left(n^{\frac{1}{1+\varepsilon}}\log n+|\mathcal{NN}_P(q,r,\varepsilon)|\log n\right)$ time using $O\left(n+n^{1+\frac{1}{1+\varepsilon}}\log n\right)$ space.*

See Appendix A for the proof. As expected, our query time depends on two terms: the first term is sub-linear in the input size and corresponds to the complexity of a standard (r,ε)-$\mathcal{NN}$ query. It can be viewed as the incompressible time needed to locate the query point q in our data structure. The second term arises from the fact that our answer is a set instead of a single point. Ideally, one would like it to be the size of $\mathcal{NN}_P(q,r)$. However, since we only produce an approximating superset of $\mathcal{NN}_P(q,r)$, our second term rises up to $|\mathcal{NN}_P(q,r,\varepsilon)|$ times a logarithmic factor due to our design.

3 The static ε-$\mathcal{RNN}$ problem

In this section we show how the data structures of Section 2 can be used to solve the ε-$\mathcal{RNN}$ problem in a static setting. In fact, our approach uses the data structures of Section 2 as black boxes, so it can be easily made dynamic by replacing them by their dynamic analogs, as will be discussed in Section 5.

We divide our procedure into two phases: the pre-computation phase where we build the data structure, and the online query phase where we find the approximate reverse nearest neighbors of a given query point. The full procedure is described in Section 3.2, its output proven correct in Section 3.3, and its complexity analyzed in Section 3.4. We begin with an overview of the approach and of its key ingredients in Section 3.1.

3.1 Overview of the approach

Suppose the distance of every blue point $x\in B$ to the red point set R has been pre-computed. Then, given a query point q and a parameter ε, computing a solution to the ε-$\mathcal{RNN}$ query

amounts to checking, for each point $x \in B$, whether $\mathrm{d}(q,x) \leq d_{\mathcal{NN}_R}(x)$, or $\mathrm{d}(q,x) > (1+\varepsilon)d_{\mathcal{NN}_R}(x)$, or $d_{\mathcal{NN}_R}(x) < \mathrm{d}(q,x) \leq (1+\varepsilon)d_{\mathcal{NN}_R}(x)$: in the first case, x must be included in the solution; in the second case, it must not; and in the third case, it can be indifferently inserted or not. Performing this check can be done by computing a solution S to the all-(r,ε)-$\mathcal{NN}$ query described in Problem 4, with $P = B$ and $r = d_{\mathcal{NN}_R}(x)$, and by including x in the answer if and only if it belongs to S. Indeed,

Observation 1. $x \in \mathcal{RNN}_{B,R}(q) \Rightarrow x \in S \Rightarrow x \in \mathcal{RNN}_{B,R}(q,\varepsilon)$.

Proof. According to Problem 4, we have $\mathcal{NN}_B(q, d_{\mathcal{NN}_R}(x), 0) \subseteq S \subseteq \mathcal{NN}_B(q, d_{\mathcal{NN}_R}(x), \varepsilon)$. Therefore, $x \in \mathcal{RNN}_{B,R}(q) \Rightarrow \mathrm{d}(x,q) \leq d_{\mathcal{NN}_R}(x) \Rightarrow x \in \mathcal{NN}_B(q, d_{\mathcal{NN}_R}(x), 0) \Rightarrow x \in S \Rightarrow x \in \mathcal{NN}_B(q, d_{\mathcal{NN}_R}(x), \varepsilon) \Rightarrow \mathrm{d}(x,q) \leq (1+\varepsilon)d_{\mathcal{NN}_R}(x) \Rightarrow x \in \mathcal{RNN}_{B,R}(q,\varepsilon)$. □

The problem with this approach is that parameter r varies with the blue point x considered, so the total number of all-(r,ε)-$\mathcal{NN}$ queries to be solved can be up to linear. To reduce this number we use a bucketing strategy. In a pre-processing phase, we compute $d_{\mathcal{NN}_R}(x)$ for every point $x \in B$ and then hash the blue points into buckets according to their nearest neighbor distances, so that bucket B_i contains every $x \in B$ such that $d_{\mathcal{NN}_R}(x) \in \big[(1+\varepsilon)^{i-1}, (1+\varepsilon)^i\big)$. At query time, we solve the all-(r,ε)-$\mathcal{NN}$ problem with $r = (1+\varepsilon)^i$ on each bucket B_i separately, and then we take the union of the solutions as our output. Since the points $x \in B_i$ satisfy $(1+\varepsilon)^{i-1} \leq d_{\mathcal{NN}_R}(x) < (1+\varepsilon)^i$, Observation 1 guarantees that our output is an admissible solution to the $\varepsilon(2+\varepsilon)$-$\mathcal{RNN}$ query on q. Constants in the detailed algorithm description will be adapted so as to make the output an admissible solution to the ε-$\mathcal{RNN}$ query on q.

Note that we do not impose any constraints on parameter i. This means that we have to inspect every non-empty bucket B_i at query time. As a result, in pathological cases such as when all non-empty buckets are singletons, we will end up considering a linear number of buckets, even though $\mathcal{RNN}_{B,R}(q,\varepsilon)$ itself may be empty. To avoid this pitfall, we need to limit the range of values of i to be considered. This is done by exploiting the following observations, where y is an arbitrary point of $\mathcal{NN}_R(q,\varepsilon)$:

Observation 2. *Every point $x \in \mathcal{RNN}_{B,R}(q)$ is such that $d_{\mathcal{NN}_R}(x) \geq \frac{\mathrm{d}(q,y)}{2(1+\varepsilon)}$.*

Proof. Let $x \in B$ be such that $d_{\mathcal{NN}_R}(x) < \frac{1}{2(1+\varepsilon)}\mathrm{d}(q,y)$, and let $p \in R$ be closest to x. Then, $\mathrm{d}(p,x) = d_{\mathcal{NN}_R}(x) < \frac{1}{2(1+\varepsilon)}\mathrm{d}(q,y)$, which is at most $\frac{1}{2}\mathrm{d}(q,p)$ since $y \in \mathcal{NN}_R(q,\varepsilon)$. It follows that $\mathrm{d}(p,x) < \mathrm{d}(q,x)$, which means that $x \notin \mathcal{RNN}_{B,R}(q)$. □

Observation 3. *Every point $x \in \mathcal{RNN}_{B,R}(q)$ such that $d_{\mathcal{NN}_R}(x) \geq \frac{2\mathrm{d}(q,y)}{\varepsilon}$ belongs to $\mathcal{RNN}_{B,R}(y,\varepsilon/2)$.*

Proof. Since $x \in \mathcal{RNN}_{B,R}(q)$, we have $\mathrm{d}(x,q) = d_{\mathcal{NN}_{R\cup\{q\}}}(x) \leq d_{\mathcal{NN}_R}(x)$. In addition, we have $\mathrm{d}(q,y) \leq \frac{\varepsilon}{2}d_{\mathcal{NN}_R}(x)$ by hypothesis. It follows that $\mathrm{d}(x,y) \leq \mathrm{d}(x,q) + \mathrm{d}(q,y) \leq (1+\frac{\varepsilon}{2})d_{\mathcal{NN}_R}(x)$, which means that x belongs to $\mathcal{RNN}_{B,R}(y,\varepsilon/2)$. □

Assuming that we have precomputed a data structure that enables us to find some $y \in \mathcal{NN}_R(q,\varepsilon)$, Observation 2 guarantees that we can safely ignore those buckets B_i with $i < \log_{1+\varepsilon}\frac{\mathrm{d}(q,y)}{2(1+\varepsilon)}$. Furthermore, assuming that the set $\mathcal{RNN}_{B,R}(y,\varepsilon/2)$ has been precomputed, Observation 3 guarantees that the reverse nearest neighbors of q that belong to those buckets B_i with $i \geq \log_{1+\varepsilon}\frac{2\mathrm{d}(q,y)}{\varepsilon}$ can be looked for among the points of $\mathcal{RNN}_B(y,\varepsilon/2)$. Thus, the total number of buckets to be inspected is reduced to $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$.

3.2 Details of the data structure and algorithms

Given a metric space (X, d), two point sets $B, R \subseteq X$ (referred to as blue and red points respectively), a parameter $\varepsilon \geq 0$, and a locality-sensitive family $\mathcal{F}$ of hash functions, the pre-computation phase builds a data structure $\mathcal{RNNDS}(B, R, \varepsilon)$ that stores the following three pieces of information (see Algorithm 1):

- A collection of buckets $\{B_i\}_{i\in\mathbb{Z}}$ that partitions B. Only the non-empty buckets are stored, and in a hash-table that ensures constant look-up time. For each i, the points $x \in B$ such that $(1+\varepsilon)^{(i-1)/3} \leq d_{\mathcal{NN}_R}(x) < (1+\varepsilon)^{i/3}$ are distributed among B_i and B_{i+1} in a way that is made precise on lines 2-5 of Algorithm 1. For each bucket B_i we build the modified LSH data structure $\mathcal{A}(B_i, (1+\varepsilon)^{i/3}, (1+\varepsilon)^{1/3}-1)$ introduced in Section 2.4, in order to be able to answer all-$((1+\varepsilon)^{i/3}, (1+\varepsilon)^{1/3}-1)$-$\mathcal{NN}$ queries on the set B_i (lines 6-8).
- For each red point y, a set B_y of blue points that contains $\mathcal{RNN}_{B,R}(y, \varepsilon/2)$ and that is small enough to be contained in $\mathcal{RNN}_{B,R}(y, \varepsilon)$ (lines 9-12).
- The search tree $\mathcal{T}(R, \varepsilon)$ and its associated LSH data structures, in order to be able to answer ε-$\mathcal{NN}$ queries on R (line 13). In dynamic settings, $\mathcal{T}(R, \varepsilon)$ can be replaced by the data structure introduced in Section 4 below.

Now, given the $\mathcal{RNNDS}(B, R, \varepsilon)$ data structure and a query point q, we answer the ε-$\mathcal{RNN}$ query as follows (see Algorithm 2):

- We find an ε-approximate nearest neighbor y of q among the red point set R (line 1).
- We solve an approximate all-nearest neighbors query on each bucket B_i separately, for i in the range prescribed by Observations 2 and 3 (lines 2-6), and then we gather the solutions in a set S (line 7).
- We inspect the points of B_y, and add to S the ones that are far enough from y (lines 8-12).

Upon termination, we return the set S (line 13).

Input : metric space (X, d), point sets $B, R \subseteq X$, parameter $\varepsilon \geq 0$, integer c, LSH family $\mathcal{F}$
Output: $\mathcal{RNNDS}(B, R, \varepsilon)$ data structure

1 Initialize $B_i = \emptyset$ for $i \in \mathbb{Z}$
2 **foreach** $x \in B$ **do**
3 | Compute $y_x \in \mathcal{NN}_R(x, (1+\varepsilon)^{1/3}-1)$ *// implemented by brute-force in static settings, // by using LSH data structures in dynamic settings*
4 | Find i s.t. $(1+\varepsilon)^{(i-1)/3} \leq \mathrm{d}(x, y_x) < (1+\varepsilon)^{i/3}$ and update $B_i = B_i \cup \{x\}$
5 **end**
6 **foreach** $B_i \neq \emptyset$ **do**
7 | Build $\mathcal{A}(B_i, (1+\varepsilon)^{i/3}, (1+\varepsilon)^{1/3}-1)$ data structure using parameter c and LSH family $\mathcal{F}$
8 **end**
9 **foreach** $y \in R$ **do**
10 | Compute $B_y \subseteq B$ such that $\mathcal{RNN}_{B,R}(y, \varepsilon/2) \subseteq B_y \subseteq \mathcal{RNN}_{B,R}(y, \varepsilon)$ *// implemented by brute force in static settings, // by using $\mathcal{RNNDS}$ data structures in dynamic settings*
11 | Sort the points of B_y according to their distances to y
12 **end**
13 Build $\mathcal{T}(R, \varepsilon)$ and associated LSH data structures *// to be replaced by the data structure of Section 4 in dynamic settings*

Algorithm 1: Pre-processing phase of ε-$\mathcal{RNN}$.

Input : metric space (X, d), $\mathcal{RNNDS}(B, R, \varepsilon)$ data structure, query point $q \in X$

1 Use $\mathcal{T}(R, \varepsilon)$ and LSH data structures to answer ε-$\mathcal{NN}$ query on input (R, q); let $y \in R$ be the output // $y \in \mathcal{NN}_R(q, \varepsilon)$ *with high probability, by Corollary 2.4*
2 **for** $i = \left\lfloor 3 \log_{1+\varepsilon} \frac{\mathrm{d}(q,y)}{2(1+\varepsilon)} \right\rfloor$ to $\left\lceil 3 \log_{1+\varepsilon} \frac{2\mathrm{d}(q,y)}{\varepsilon} \right\rceil + 2$ **do**
3 **if** $B_i \neq \emptyset$ **then**
4 Solve all-$((1+\varepsilon)^{i/3}, (1+\varepsilon)^{1/3} - 1)$-$\mathcal{NN}$ query on input (B_i, q); let S_i be the output
5 **end**
6 **end**
7 Let $S = \bigcup_i S_i$
8 **foreach** $x \in B_y$ such that $\mathrm{d}(x, y) \geq \frac{2(1+\varepsilon)}{\varepsilon} \mathrm{d}(q, y)$ **do**
9 **if** $\mathrm{d}(x, q) \leq \mathrm{d}(x, y)$ **then**
10 $S = S \cup \{x\}$
11 **end**
12 **end**
13 Return S

Algorithm 2: Query phase of ε-$\mathcal{RNN}$.

3.3 Correctness of the output

For clarity, we let S_1 be the set of points inserted in S during steps 2 through 7 of Algorithm 2, and S_2 be the set of points inserted in S during steps 8 through 12. The output of the algorithm is $S_1 \cup S_2$. Our plan is to show that S_1 is correct with high probability (Lemma 3.1), then that S_2 is also correct with high probability (Lemma 3.2), from which we will deduce that the output $S_1 \cup S_2$ itself is correct with high probability (Theorem 3.3). Let $B_{S_1} = \bigcup_i B_i$ for $i = \left\lfloor 3 \log_{1+\varepsilon} \frac{\mathrm{d}(q,y)}{2(1+\varepsilon)} \right\rfloor$ to $\left\lceil 3 \log_{1+\varepsilon} \frac{2\mathrm{d}(q,y)}{\varepsilon} \right\rceil + 2$.

Lemma 3.1. *With high probability,* $\mathcal{RNN}_{B,R}(q) \cap B_{S_1} \subseteq S_1 \subseteq \mathcal{RNN}_{B,R}(q, \varepsilon)$.

Proof. Recall from line 7 of Algorithm 2 that S_1 is built by taking the union of the sets S_i generated by solving all-$((1+\varepsilon)^{i/3}, (1+\varepsilon)^{1/3} - 1)$-$\mathcal{NN}$ queries on the point sets B_i with query point q. Recall now from lines 3-4 of Algorithm 1 that every point $x \in B_i$ satisfies $(1+\varepsilon)^{(i-1)/3} \leq \mathrm{d}(x, y_x) < (1+\varepsilon)^{i/3}$ for some $y_x \in \mathcal{NN}_R(x, (1+\varepsilon)^{1/3} - 1)$, which implies that $(1+\varepsilon)^{(i-2)/3} \leq d_{\mathcal{NN}_R}(x) < (1+\varepsilon)^{i/3}$. By Theorem 2.5, with high probability the points $x \in B_i$ that belong to S_i also belong to $\mathcal{NN}_{B_i}(q, (1+\varepsilon)^{i/3}, (1+\varepsilon)^{1/3} - 1)$ and therefore satisfy $\mathrm{d}(x, q) \leq (1+\varepsilon)^{(i+1)/3}$, which implies that $\mathrm{d}(x, q) \leq (1+\varepsilon) d_{\mathcal{NN}_R}(x)$. Thus, $S_i \subseteq \mathcal{RNN}_{B,R}(q, \varepsilon)$. In addition, the points $x \in \mathcal{RNN}_{B,R}(q) \cap B_i$ satisfy $\mathrm{d}(x, q) \leq d_{\mathcal{NN}_R}(x) \leq (1+\varepsilon)^{i/3}$ and therefore belong to $\mathcal{NN}_{B_i}(q, (1+\varepsilon)^{i/3}, 0)$. By Theorem 2.5, this set is included in S_i with high probability, hence we have $\mathcal{RNN}_{B,R}(q) \cap B_i \subseteq S_i$.

Since the above inclusions hold w.h.p. for each non-empty set B_i taken separately, and since the total number of non-empty sets B_i is at most the number of points, we conclude by the union bound that $\mathcal{RNN}_{B,R}(q) \cap B_{S_1} \subseteq S_1 \subseteq \mathcal{RNN}_{B,R}(q, \varepsilon)$ with high probability, where B_{S_1} is the union of the B_i and S_1 is the union of the S_i for i ranging from $\left\lfloor 3 \log_{1+\varepsilon} \frac{\mathrm{d}(q,y)}{2(1+\varepsilon)} \right\rfloor$ to $\left\lceil 3 \log_{1+\varepsilon} \frac{2\mathrm{d}(q,y)}{\varepsilon} \right\rceil + 2$. □

Lemma 3.2. *With high probability,* $\mathcal{RNN}_{B,R}(q) \setminus B_{S_1} \subseteq S_2 \subseteq \mathcal{RNN}_{B,R}(q, \varepsilon)$.

Proof. The first inclusion follows directly from Observations 2 and 3. Indeed, recall from the proof of Lemma 3.1 that every point $x \in B_i$ with $i < \left\lfloor 3\log_{1+\varepsilon} \frac{\mathrm{d}(q,y)}{2(1+\varepsilon)} \right\rfloor$ satisfies $d_{\mathcal{NN}_R}(x) < (1+\varepsilon)^{i/3} < \frac{\mathrm{d}(q,y)}{2(1+\varepsilon)}$ and therefore cannot belong to $\mathcal{RNN}_{B,R}(q)$, by Observation 2. This conclusion only holds provided that $y \in \mathcal{NN}_R(q,\varepsilon)$, which is true with high probability according to line 1 of Algorithm 2. In addition, every point $x \in \mathcal{RNN}_{B,R}(q) \cap B_i$ with $i > \left\lceil 3\log_{1+\varepsilon} \frac{2\mathrm{d}(q,y)}{\varepsilon} \right\rceil + 2$ satisfies $d_{\mathcal{NN}_R}(x) \geq (1+\varepsilon)^{(i-2)/3} > \frac{2\mathrm{d}(q,y)}{\varepsilon}$ and therefore belongs to $\mathcal{RNN}_{B,R}(y,\varepsilon/2)$, by Observation 3 (here again, the conclusion only holds provided that $y \in \mathcal{NN}_R(q,\varepsilon)$, which is true with high probability). Since by construction $\mathcal{RNN}_{B,R}(y,\varepsilon/2) \subseteq B_y$, all such points x are considered in the loop of lines 8-12 of Algorithm 2 and are therefore inserted in S. It follows that $\mathcal{RNN}_{B,R}(q) \setminus B_{S_1} \subseteq S_2$.

The second inclusion is easy: since by construction we have $S_2 \subseteq B_y$ (lines 8-12 of Algorithm 2) and $B_y \subseteq \mathcal{RNN}_{B,R}(y,\varepsilon)$ (line 10 of Algorithm 1), every point $x \in S_2$ satisfies $\mathrm{d}(x,q) \leq \mathrm{d}(x,y) \leq (1+\varepsilon)d_{\mathcal{NN}_R}(x)$, which means that $x \in \mathcal{RNN}_{B,R}(q,\varepsilon)$. Hence, $S_2 \subseteq \mathcal{RNN}_{B,R}(q,\varepsilon)$. □

Our correctness guarantee follows directly from Lemmas 3.1 and 3.2, using the union bound:

Theorem 3.3. *Given a query point $q \in X$, Algorithm 2 outputs a point set $S_1 \cup S_2$ such that $\mathcal{RNN}_{B,R}(q) \subseteq S_1 \cup S_2 \subseteq \mathcal{RNN}_{B,R}(q,\varepsilon)$ with high probability.*

Note that our analysis relies on two important prerequisites:
(a) that y_x belongs to $\mathcal{NN}_R(x,(1+\varepsilon)^{1/3}-1)$ on line 3 of Algorithm 1, and
(b) that B_y is sandwiched between $\mathcal{RNN}_{B,R}(y,\varepsilon/2)$ and $\mathcal{RNN}_{B,R}(y,\varepsilon)$ on line 10 of Algorithm 1.

These conditions are trivially satisfied in static settings, where lines 3 and 10 are implemented by brute-force search. In dynamic settings however, they would need to be enforced through a clever choice of parameters in the data structures of Section 2.

3.4 Complexity

Theorem 3.4. *The running time of Algorithm 2 is $O\left(\frac{1}{\varepsilon}n^{\frac{1}{1+\varepsilon}}\log\frac{n}{\varepsilon}\log n + |\mathcal{RNN}_{B,R}(q,O(\varepsilon))|\log n\right)$, and the size of the $\mathcal{RNNDS}(B,R,\varepsilon)$ data structure is $O\left(\frac{1}{\varepsilon}n^{1+\frac{1}{(1+\varepsilon)^{1/3}}}\mathrm{polylog}\frac{n}{\varepsilon} + \sum_{y\in R}|B_y|\right)$.*

Note that the first term in the size bound is always sub-quadratic, whereas the second term varies between linear (when $B_y = \mathcal{RNN}_{B,R}(y)$ for all $y \in R$) and quadratic.

Proof of Theorem 3.4. First we bound the query time, then we bound the size of the data structure:

Bound on the query time: Algorithm 5 consists of three steps (line 1, lines 2-7, lines 8-12) that can be analyzed separately:

- The ε-$\mathcal{NN}$ query on line 1 takes $O\left(\frac{1}{\varepsilon}n^{\frac{1}{1+\varepsilon}}\log\frac{n}{\varepsilon}\log n\right)$ time, by Corollary 2.4.
- The total number of buckets considered in the loop of lines 2-6 is $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$. For each non-empty bucket B_i, the all-$\mathcal{NN}$ query takes $O\left(n^{\frac{1}{1+\varepsilon}}\log n + |\mathcal{NN}_{B_i}(q,(1+\varepsilon)^{i/3},(1+\varepsilon)^{1/3}-1)|\log n\right)$ time, by Theorem 2.5. Now, as pointed out in the proof of Lemma 3.1, the points $x \in \mathcal{NN}_{B_i}(q,(1+\varepsilon)^{i/3},(1+\varepsilon)^{1/3}-1)$ satisfy $\mathrm{d}(x,q) \leq (1+\varepsilon)^{(i+1)/3} \leq (1+\varepsilon)d_{\mathcal{NN}_R}(x)$, which means that $\mathcal{NN}_{B_i}(q,(1+\varepsilon)^{i/3},(1+\varepsilon)^{1/3}-1) \subseteq \mathcal{RNN}_{B,R}(q.\varepsilon)$. Since the buckets

B_i are pairwise disjoint, so are the sets $\mathcal{NN}_{B_i}(q,(1+\varepsilon)^{i/3},(1+\varepsilon)^{1/3}-1)$, and therefore the total running time of lines 2-7 is $O\left(\frac{1}{\varepsilon}n^{\frac{1}{1+\varepsilon}}\log n\log\frac{1}{\varepsilon}+|\mathcal{RNN}_{B,R}(q,\varepsilon)|\log n\right)$.

- Since the points of B_y have been sorted according to their distances to y (line 11 of Algorithm 1), looking up for the subset of the points $x\in B_y$ such that $\mathrm{d}(x,y)\geq\frac{2(1+\varepsilon)}{\varepsilon}\mathrm{d}(q,y)$ takes $O(\log|B_y|)=O(\log n)$ time. Now, for each such point x, we have $\mathrm{d}(x,q)\leq\mathrm{d}(x,y)+\mathrm{d}(y,q)\leq(1+\frac{\varepsilon}{2(1+\varepsilon)})\mathrm{d}(x,y)$, which is at most $(1+\varepsilon)(1+\frac{\varepsilon}{2(1+\varepsilon)})d_{\mathcal{NN}_R}(x)$ since by construction $B_y\subseteq\mathcal{RNN}_{B,R}(y,\varepsilon)$. As a result, the total number of points considered in the loop of lines 8-12 of Algorithm 2 is at most $|\mathcal{RNN}_{B,R}(q,\varepsilon(1+\frac{1}{2(1+\varepsilon)}+\frac{\varepsilon}{2(1+\varepsilon)}))|$.

Combining these bounds, we obtain the claimed upper bound on the query time.

Bound on the size: As mentioned in Section 3.2, the $\mathcal{RNNDS}(B,R,\varepsilon)$ data structure consists of a collection of pairwise-disjoint non-empty buckets, of total cardinality at most n, and for each bucket B_i a $\mathcal{A}(B_i,(1+\varepsilon)^{i/3},(1+\varepsilon)^{1/3}-1)$ data structure of size $O\left(n_i^{1+\frac{1}{(1+\varepsilon)^{1/3}}}\log n_i\right)$ where $n_i=|B_i|$ (Theorem 2.5). This gives a total size of $O(\sum_i n_i^{1+\frac{1}{(1+\varepsilon)^{1/3}}}\log n_i)=O(n^{1+\frac{1}{(1+\varepsilon)^{1/3}}}\log n)$. In addition, $\mathcal{RNNDS}(B,R,\varepsilon)$ stores the search tree $\mathcal{T}(R,\varepsilon)$ and its associated LSH data structures, whose total size is $O\left(\frac{1}{\varepsilon}n^{1+\frac{1}{1+\varepsilon}}\mathrm{polylog}\frac{n}{\varepsilon}\right)$, by Corollary 2.4. Finally, the data structure stores the set B_y for each point $y\in R$, which makes a total size of $O(\sum_{y\in R}|B_y|)$. Combining the above bounds, we obtain the claimed upper bound on the size of the data structure. □

4 Dynamic search trees in affine metric spaces

In this section we show how the tree structure of Har-Peled [17] can be made dynamic when the metric space considered is affine. Recall from Section 2.2 that each node v of $\mathcal{T}(P,\varepsilon)$ is associated with a subset P_v of P and with an interval $[r_v,R_v]$ of admissible distances, so that every ε-$\mathcal{NN}$ query can be answered by traversing down the tree according to the following rules (applied at each visited node v):

- If the query point q satisfies $d_{\mathcal{NN}_P}(q)\in[r_v,R_v]$, then a point of $\mathcal{NN}_P(q,\varepsilon)$ can be found at node v without recursing down the tree by answering $\log_{1+\varepsilon}\frac{R_v}{r_v}$ queries of type (r,ε)-$\mathcal{NN}$ using the LSH structure from Section 2.3 with geometrically increasing r.
- Otherwise, one needs to recurse down into exactly one of the children of v: an *inner child* when $d_{\mathcal{NN}_P}(q)<r_v$, the *outer child* when $d_{\mathcal{NN}_P}(q)>R_v$ (see below for the definitions). In the former case, the inner child into which to recurse is the one containing the output of the (r_v,ε)-$\mathcal{NN}$ query.

Remark 1. *Our description follows [18, §14.3] and ignores the fact that the* (r_v,ε)-$\mathcal{NN}$ *and* (R_v,ε)-$\mathcal{RNN}$ *queries only tell us* approximately *whether* $d_{\mathcal{NN}_P}(q)<r_v$ *or* $r_v\leq d_{\mathcal{NN}_P}(q)\leq R_v$ *or* $R_v<d_{\mathcal{NN}_P}(q)$*. As shown in [18, §14.3], the parameters of the data structure can be slightly modified and the analysis adapted to account for this fact, at the price of increased technicality. We will therefore continue to ignore it, in order to simplify the exposition.*

The construction of the tree $\mathcal{T}(P,\varepsilon)$ is done in a top-down fashion, starting with the root (with which the entire point set P is associated) and ending with the leaves (with which subsets of P of constant size are associated, so the ε-$\mathcal{NN}$ query can be answered quickly by brute-force search). Once a node v has been created and a subset of points $P_v\subseteq P$ associated with it, its children are defined as follows: call P_v^r the union of the closed balls of same radius r about the points of P_v. Let r_v be the minimum value of r such that this union has $|P_v|/2$ connected components. Each connected component C_i of $P_v^{r_v}$ gives rise to an *inner child* v_i of v, to which is associated the set P_{v_i} of the ball centers lying in C_i. Thus, $P_{v_1},\cdots,P_{v_k}$ partition P_v according to the connected

components of the union of balls $P_v^{r_v}$. In addition, one point p_{v_i} is chosen arbitrarily in each set P_{v_i} to represent it, and the points p_{v_i} are gathered together to form a subset Q of P_v that gives rise to another child v_0 of v, called the *outer child*. Letting $R_v = 8\frac{|P_v|\lceil\log_{3/2} n\rceil}{\varepsilon} r_v$, where $n = |P|$, this construction guarantees that the following conditions are met (with $c = c' = 1/2$):

Property 1 (see Chapter 14 of [18]). *$\exists c, c' \in (0,1)$ such that, at every node v of $\mathcal{T}(P,\varepsilon)$,*

(i) *$P_{v_1}, \ldots, P_{v_k}$ form a partition of P_v,*
(ii) *$k \leq c|P_v|$ and $\forall i$, $|P_{v_i}| \leq c'|P_v|$,*
(iii) *$\forall i$, the diameter of P_{v_i} is at most $\frac{\varepsilon}{4\lceil\log_{3/2} n\rceil} R_v$,*
(iv) *$\forall i \neq j$, $\forall p_i \in P_i$, $\forall p_j \in P_j$, $\mathrm{d}(p_i, p_j) > 2r_v$,*
(v) *$\frac{R_v}{r_v}$ is at most a polynomial in $\frac{n}{\varepsilon}$ of bounded degree, so $\log_{1+\varepsilon} \frac{R_v}{r_v} = O(\frac{1}{\varepsilon}\log\frac{n}{\varepsilon})$.*

Har-Peled's analysis relies exclusively on the above set of conditions. More precisely, Condition (i) guarantees that P_v and $\bigcup_{i=1}^k P_{v_i}$ represent the same point cloud, with balls of different radii. Condition (ii) guarantees that the height of the tree $\mathcal{T}(P,\varepsilon)$ is logarithmic in n. Condition (iii) guarantees that whenever a query point q satisfies $d_{\mathcal{NN}_{P_v}}(q) > R_v$, we can recurse into the outer child v_0, where point p_{v_i} can act as a representative of the cluster P_{v_i} since every point of P_{v_i} belongs to $\mathcal{NN}_{P_{v_i}}(q, \frac{\varepsilon}{4\lceil\log_{3/2} n\rceil})$ (see Lemma 14.2.9(ii) of Har-Peled [18]). Condition (iv) guarantees that the clusters are sufficiently far apart from one another, so the inner child v_i into which to recurse is clearly identified when $d_{\mathcal{NN}_{P_v}}(q) < r_v$. Finally, Condition (v) guarantees that whenever $r_v \leq d_{\mathcal{NN}_{P_v}}(q) \leq R_v$, the number of (r,ε)-$\mathcal{NN}$ queries needed to locate $d_{\mathcal{NN}_{P_v}}(q)$ within the interval is logarithmic in $\frac{n}{\varepsilon}$.

The clustering of P_v obtained from the connected components of a union of balls is very fragile by nature, since an insertion can change the connectivity of the union drastically and thus merge a potentially high number of nodes. In particular, one cannot hope to prevent such situations to occur under an adversarial insertion/deletion model, where repairing the tree after each insertion/deletion could become very costly. To work around this issue we simply change the clustering strategy, using a random grid decomposition instead of the deterministic union of balls decomposition. The rest of the procedure remains exactly the same[4], the only difference being the exact shape of the tree. While grid decompositions require the ambient space to be affine (in the sequel we will take $(\mathbb{R}^d, \ell_2)$ for simplicity), they are easier to maintain under point insertions and deletions. We re-emphasize that a deterministic structure cannot maintain Property 1 in an adversarial setting where an adversary is performing the insertions and deletions. Hence, our grids will be randomized, and our adversary will be assumed oblivious.

4.1 Construction of the search tree

In a pre-computation phase, we build our search tree in a top-down fashion, as in [18, Chapter 14]. At each newly created node v, we partition the set P_v according to the cells of the random grid built by the `CreateGrid` algorithm. This algorithm chooses a random origin o and step size ℓ, and builds the corresponding regular axis-aligned grid. This procedure is repeated until the grid cells contain at most $3m/4$ points of P_v each, where $m = |P_v|$. Then, the algorithm repeatedly merges neighboring grid cells in a particular order (see Algorithm 3 for the details), until there remain at most $3m/4$ non-empty cells. The output is a semi-regular grid of the form $2^{a_1}\ell\mathbb{Z} \times 2^{a_2}\ell\mathbb{Z} \times \cdots \times 2^{a_d}\ell\mathbb{Z}$, where $a_1, a_2, \cdots, a_d$ are integers, such that at most $3m/4$ cells are non-empty and contain at most $3m/4$ points of P_v each. The intersections of these cells with P_v define the clusters $P_{v_1}, \cdots, P_{v_k}$. We also let $r_v = \frac{\ell}{104dn^2}$ and $R_v = \frac{936\lceil\log_{3/2} n\rceil d\sqrt{d}}{\varepsilon}\ell$.

Quality of the output.

[4] In particular, our query procedure traverses down the tree exactly the same way as in [17].

Input : A finite point set $P_v \subseteq \mathbb{R}^d$ of size $|P_v| = m$
Output: Clusters $P_{v_1}, \cdots, P_{v_k}$ that partition P_v, and a set of representative points $p_{v_1} \in P_1, \cdots, p_{v_k} \in P_k$

1 **repeat**
2 | Choose x and y uniformly at random from P_v
3 | Let $\ell = \frac{\|x-y\|_1}{d}$
4 | Let $a = \lceil \log_2(117d) \rceil$
5 | Choose a point o uniformly at random from $[0, 2^a\ell)^d$
6 | Let G be the regular axis-aligned grid of step size ℓ and origin o
7 **until** *all grid cells have* $\leq \frac{3m}{4}$ *points*
8 **while** *there are more than* $\frac{3m}{4}$ *non-empty grid cells* **do**
9 | Pick a Cartesian axis along which the step size of G is minimal
10 | Merge every other cell with its predecessor along this axis, while maintaining o as a vertex
11 | **if** *the step size along some Cartesian axis is more than* $2^a\ell$ **then**
12 | | Discard grid G and go to line 1
13 | **end**
14 **end**
15 Let $C_1, \cdots, C_k$ be the non-empty cells of G
16 **for** $i = 1$ to k **do**
17 | Let $P_{v_i} = P_v \cap C_i$
18 **end**
19 Return $P_{v_1}, \cdots P_{v_k}$, and any set of points $p_{v_1}, \cdots p_{v_k}$, where $p_{v_i} \in P_i$

Algorithm 3: The `CreateGrid` algorithm

Theorem 4.1. *The output of Algorithm 3 satisfies the conditions of Property 1 with high probability.*

Thus, by plugging our grid decomposition in place of the union of balls decomposition in Har-Peled's tree structure (and analysis), we obtain a data structure that can answer static ε-$\mathcal{NN}$ queries in sub-linear time.

Proof of Theorem 4.1. We check the conditions of Property 1 one by one:
(i) This condition is trivially satisfied by construction.
(ii) The test on line 8 guarantees that $k \leq 3m/4$. If in addition the algorithm terminates without entering the *while* loop, then the test on line 7 ensures that every cell of the output grid G contains at most $3m/4$ points of P_v. Otherwise, let G' be the grid considered at the beginning of the last execution of the *while* loop. Since the test on line 8 failed with G', the latter has more than $3m/4$ non-empty grid cells, and so no cell of G' can contain more than $m/4$ points of P_v. Since G is obtained from G' by merging disjoint pairs of cells, no cell of G can contain more than $m/2$ points of P_v. Hence, condition (ii) is satisfied with $c = c' = 3/4$.
(iii) The test on line 11 ensures that each grid cell has diameter at most $2^a\ell\sqrt{d} \leq 234\ell d\sqrt{d}$, which by definition of R_v is at most $\frac{\varepsilon}{4\lceil \log_{3/2} n \rceil} R_v$. The same bound holds for the cluster diameters by construction.
(iv) Lemma 4.5 below ensures that this condition holds with high probability.
(v) By definition, we have $\frac{R_v}{r_v} = \frac{97344n^2\lceil \log_{3/2} n \rceil d^2\sqrt{d}}{\varepsilon}$, which is at most a polynomial in n/ε of bounded degree (ignoring the polynomial dependence on d). □

We now give the details of the proof of condition (iv). Let $j \leq \binom{m}{2}$ be such that $\ell d = \|x-y\|_1$ is the j^{th} smallest ℓ_1-distance between pairs of points in P_v. We will say that ℓ is *well chosen* when $\frac{9m^2}{58} \leq j \leq \frac{9m^2}{32} - \frac{3m}{8}$.

Lemma 4.2. *If $m \geq 378$, then at the end of line 3, ℓ is well chosen with probability at least $\frac{1}{4}$.*

Proof. The probability of ℓ being well chosen is precisely

$$\frac{1}{\binom{m}{2}}\left(\left\lfloor \frac{9m^2}{32} - \frac{3m}{8} \right\rfloor - \left\lceil \frac{9m^2}{58} \right\rceil + 1\right) \geq \frac{0.126m^2 - 0.375m - 1}{0.5m^2},$$

which is at least $\frac{1}{4}$ for $m \geq 378$. □

Lemma 4.3. *If ℓ is well chosen, then every grid cell contains at most $3m/4$ points. As a result, Algorithm 3 exits the* `repeat` *loop at line 7.*

Proof. Assume that some cell of the grid contains more than $\frac{3m}{4}$ points from P_v. Since these points are all contained within a single cell, they are separated from one another by an ℓ_1-distance less than $d\ell = \|x-y\|_1$. This means that there are at least $\binom{3m/4}{2}$ pairs of points in P_v with an ℓ_1-distance less than[5] $\|x-y\|_1$. Hence, $j > \frac{1}{2} \times \frac{3m}{4} \times \left(\frac{3m}{4} - 1\right) = \frac{9m^2}{32} - \frac{3m}{8}$, which means that ℓ is not well chosen, a contradiction with the hypothesis of the lemma. □

Observe that, at the end of line 10 of Algorithm 3, all grid cells have twice the volume they had before, and their lengths along different Cartesian axes differ by at most a factor of 2.

Lemma 4.4. *When ℓ is well chosen, the probability that the grid G ends up being discarded at line 12 is at most $\frac{12}{13}$.*

Proof. Suppose the grid G eventually gets discarded at line 12. Let G' be the grid considered at the end of the previous iteration of the *while* loop (lines 8 through 14 of Algorithm 3). Since G' is not discarded, its step size along any Cartesian axis is at most $2^a\ell$. In fact, since G is discarded and has step sizes within a factor of 2 of one another, G' must be a perfectly regular grid of step size $2^a\ell$. We will bound the probability that G' contains more than $3m/4$ cells, i.e. the probability that the algorithm actually stayed in the *while* loop.

For each point $p \in P_v$, let j_p denote the number of points in $P \setminus \{p\}$ that lie within ℓ_1-distance ℓd of p. Then, $\sum_{p \in P_v} j_p$ double-counts the number of pairs of points of P_v separated by an ℓ_1-distance of at most $d\ell = \|x-y\|_1$. Therefore, $\sum_{p \in P_v} j_p = 2j$, and hence there exists some point $p \in P_v$ with $j_p \geq \frac{2j}{m}$. Consider now any point $q \in P_v$ with $\|p-q\|_1 \leq \ell d$, and let p_i and q_i denote the coordinates of p and q respectively along the i-th Cartesian axis. Then, p and q end up in different cells of G' along the i-th axis with probability exactly $\min\{\frac{|p_i - q_i|}{2^a\ell}, 1\}$. Taking a union bound over the axes, we obtain that the probability that p and q lie in different cells of G' is at most $\sum_{i=1}^{d} \frac{|p_i-q_i|}{2^a\ell} \leq \frac{d\ell}{2^a\ell} < \frac{1}{117}$.

Let X be the random variable giving the number of non-empty cells in G', and let Y be the random variable giving the number of points other than p that are in the same cell of G' as p. Clearly, $X + Y \leq m$, and the previous calculation shows that $\mathrm{E}[Y] > \frac{116}{117} j_p \geq \frac{232j}{117m}$. Therefore, $\mathrm{E}[X] < m - \frac{232j}{117m}$. It follows by Markov's inequality that $X \geq \frac{13}{12}\left(m - \frac{232j}{117m}\right)$ with probability at most $\frac{12}{13}$, and hence that $X < \frac{13}{12}\left(m - \frac{232j}{117m}\right)$ with probability at least $\frac{1}{13}$. Since ℓ is well chosen, we have $j \geq \frac{9m^2}{58}$, and therefore $\frac{13}{12}\left(m - \frac{232j}{117m}\right) \leq \frac{13m}{12}\left(1 - \frac{232 \times 9}{117 \times 58}\right) = \frac{3m}{4}$. Thus, the probability that the grid G' has more than $3m/4$ non-empty cells (and therefore that the algorithm actually stayed in the *while* loop) is at most $\frac{12}{13}$. □

[5]Note that we assume every point of P_v to lie in the interior of some grid cell. This genericity condition is satisfied with probability 1 after the choice of the origin o at line 5 of the `CreateGrid` algorithm.

Using Lemmas 4.2 through 4.4, we can show that condition (iv) of Property 1 is satisfied by our grid decomposition:

Lemma 4.5. *With probability at least $1 - \frac{1}{n}$, the points of P_v lie farther than $\frac{\ell}{104dn^2}$ from the boundary of their cell in the output grid G, and therefore farther than $\frac{\ell}{52dn^2} = 2r_v$ from the other clusters.*

Proof. Let G' be the grid created on line 6 of algorithm 3 from which the output grid G is obtained by repeated merges between cells. Then, the distance of any point of P_v to its cell boundary in G is at least its distance to its cell boundary in G'. Therefore, all we need to do is to bound the latter quantity from below.

Consider a point $p \in P_v$, and call $\mathrm{d}(p, G')$ its distance to its cell boundary in G'. We want to bound the probability that $\mathrm{d}(p, G') \leq \frac{\ell}{104dn^2}$, conditioned on the fact that the output grid is derived from G'. Formally, this quantity is equal to:

$$\frac{\text{Prob}\left[\mathrm{d}(p, G') \leq \frac{\ell}{104dn^2} \text{ and } G \text{ derives from } G'\right]}{\text{Prob}\left[G \text{ derives from } G'\right]},$$

which by Lemmas 4.2 through 4.4 is at most 52 Prob $\left[\mathrm{d}(p, G') \leq \frac{\ell}{104dn^2}\right]$. We now need to bound the probability that $\mathrm{d}(p, G') \leq \frac{\ell}{104dn^2}$. Once ℓ is fixed, we are choosing a random translation for G' at line 5, or equivalently, a random translation for p. Inside every cell of G', the fraction of the volume that lies within distance $\frac{\ell}{104dn^2}$ of the cell boundary is bounded by $2d\frac{\ell}{104dn^2}\frac{\ell^{d-1}}{\ell^d} = \frac{1}{52n^2}$. Therefore, Prob $\left[\mathrm{d}(p, G') \leq \frac{\ell}{104dn^2}\right] \leq \frac{1}{52n^2}$. As a result, conditioned on the fact that the output grid is derived from G', the probability that a given point $p \in P_v$ lies within distance $\frac{\ell}{104dn^2}$ of a grid boundary is at most $\frac{1}{n^2}$. By the union bound, under the same condition, the probability that there is a point of P_v lying within distance $\frac{\ell}{104dn^2}$ of a grid boundary is at most $\frac{|P_v|}{n^2} \leq \frac{1}{n}$. □

Size and construction time of the search tree. Using Lemmas 4.2 through 4.4, we can bound the expected running time of the `CreateGrid`algorithm:

Lemma 4.6. *When P_v is large enough, the `CreateGrid` algorithm runs in expected $O(md \log d)$ time. In fact, it runs in $O(md \log d \log n)$ time with high probability.*

Proof. Lemmas 4.2 through 4.4 imply that the algorithm terminates after $O(1)$ grid choices on expectation. For each grid choice,

- only one iteration of the *repeat* loop is performed;
- it takes $O(md)$ time to project the points of P_v into cells in the termination test of line 7;
- if the *repeat* loop terminates, then at most $d\lceil \log_2(117d) \rceil$ merges are performed before either the *while* loop terminates or the grid gets discarded on line 12.
- at each iteration of the *while* loop, we only need to merge cells on line 10 and in the test of line 11, which can be done in $O(m)$ time using linked lists.

This gives a deterministic running time of $O(md \log d)$ per grid choice, and therefore a total expected running time of $O(md \log d)$.

To get the high probability result, notice that the probability for a random grid choice to avoid getting eventually discarded on line 7 or 12 is at least $\frac{1}{4} \times \frac{1}{13} = \frac{1}{52}$, by Lemmas 4.2 through 4.4. Therefore, the probability that at least one among $\log_2 n$ independent random grid choices avoids getting discarded is at least $1 - \left(\frac{51}{52}\right)^{\log_2 n} = 1 - \left(\frac{1}{n}\right)^{\log_2 \frac{52}{51}}$. □

By Lemma 4.6, at each node v of the tree our grid decomposition is computed in $O(|P_v| \log n)$ time with high probability, which by an easy recursion gives the following construction time:

Theorem 4.7. *The* `CreateGrid` *Algorithm 3 can be run recursively to build the ε-$\mathcal{NN}$ search tree (along with the appropriate LSH data structures) in $O(\frac{1}{\varepsilon}n^{1+\frac{1}{1+\varepsilon}}\text{polylog}\frac{n}{\varepsilon})$ time. The same bound holds for the size of the data structure.*

4.2 Update of the search tree under point insertions and deletions

At each insertion or deletion, we update the tree while ensuring that the conditions of Property 1 are still met. The following observations are used to guide the update process:

- Each insertion/deletion requires us to descend only into one branch of the tree at each level. This is because each insertion/deletion affects only a single cell of the grid at each level, and hence one only needs to descend into the node of the tree that is associated with that cell to recursively update the structure, resulting in a $O(\log n)$ factor for the total update time. This is in contrast with Har-Peled's data structure, where the insertion of a new ball in the union could change its connectivity drastically.
- As proved in Lemma 4.5, with high probability the insertion of a new point by an oblivious adversary does not decrease the inter-cluster distance and therefore does not violate Property 1(iv).
- Assuming that the inserted point lies far from the boundary of its grid cell, all we need to do is to insert the point into the $O(\frac{1}{\varepsilon}\log\frac{n}{\varepsilon})$ LSH structures associated with each affected node.
- In the rare cases where the insertion destroys the properties of the grid construction at some step, we rebuild the entire tree from scratch, including the grid structures at each node. The amortized cost is the cost of rebuilding the tree, times the probability that any update goes wrong. Using the union bound for the probability of error from Lemma 4.5 over the height of the tree, and the construction cost from Theorem 4.7, we see that the amortized update cost is $O(\frac{1}{\varepsilon}n^{\frac{1}{1+\varepsilon}}\text{polylog}\frac{n}{\varepsilon})$.
- Finally, we note that an update (even if it goes well) may create or destroy a cluster, hence the number of clusters might change with each update, eventually violating Property 1(ii). Since $\Omega(|P_v|)$ such updates are needed before the property is violated at some node v and the subtree rooted at v needs to be rebuilt, the amortized cost per insertion (from Theorem 4.7) is $O(\frac{1}{\varepsilon}n^{\frac{1}{1+\varepsilon}}\text{polylog}\frac{n}{\varepsilon})$.

The above observations show that for a sub-linear amortized update cost, we can maintain Property 1 under point insertions and deletions in the tree structure of section 4.1:

Theorem 4.8. *Given that each LSH data structure can be updated in amortized T_{LSH} time, the expected amortized cost of inserting or deleting a point in the tree is $O(\frac{1}{\varepsilon}(n^{\frac{1}{1+\varepsilon}}+T_{\text{LSH}})\text{polylog}\frac{n}{\varepsilon})$.*

5 Final remarks

Update of the LSH data structure. This is the last remaining piece of our analysis of the dynamic ε-$\mathcal{NN}$ problem. We only give an overview here, leaving the technical details as an exercise. First of all, we note that updating the hash tables takes sub-linear time in total, since there is only a sub-linear number k of them. However, this number k needs to be adapted when n changes. On the one hand, modifying k at each insertion or deletion results in a linear update cost. On the other hand, letting the number n of data points increase for a while before updating k decreases the chances of success of the queries, while letting n decrease for a while does not degrade the chances of success but makes the query time increase relative to n. By multiplying (resp. dividing) k by 2^ρ (see Algorithm 4) each time n has increased (resp. decreased) by a factor of 2 (resp. 1/2) since the last update of k, we can maintain the properties of the LSH data structure while guaranteeing a sub-linear amortized update cost.

Dynamic ε-$\mathcal{RNN}$ search. The main hurdle in making our approach to ε-$\mathcal{NN}$ dynamic is that the size of the $\mathcal{RNNDS}$ data structure can be up to quadratic, since each approximate reverse nearest neighbor set B_y can have up to linear size (as pointed out after Theorem 3.4). Computing exactly $\mathcal{RNN}_B(y)$ for every point $y \in R$ in a pre-processing step forces $\sum_{y \in R} |B_y|$ to be linear, and then it takes $\Omega(n)$ insertions before this quantity becomes quadratic. However, when we rebuild the data structure we can no longer afford to compute the B_y by brute-force, since this takes quadratic time in total. So there is a trade-off to be found between the cost of rebuilding and the desired quality of the approximate reverse nearest neighbor sets B_y.

References

[1] Elke Achtert, Christian Böhm, Peer Kröger, Peter Kunath, Alexey Pryakhin, and Matthias Renz. Efficient Reverse k-Nearest Neighbor Search in Arbitrary Metric Spaces. *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 515–526, 2006. 4

[2] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 459–468, 2006. doi: http://dx.doi.org/10.1109/FOCS.2006.49. 3

[3] Alexandr Andoni and Piotr Indyk. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *Communications of the ACM*, 51(1):117, 2008. 3

[4] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An Optimal Algorithm for Approximate Nearest Neighbor Searching in Fixed Dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998. 3

[5] Rimantas Benetis, Christian Jensen, Gytis Karĉiauskas, and Simonas Ŝaltenis. Nearest and Reverse Nearest Neighbor Queries for Moving Objects. *The International Journal on Very Large Data Bases*, 15(3):229–249, 2006. 4

[6] Jon L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 18(9):517, 1975. 3

[7] Jean-Daniel Boissonnat, Leonidas J. Guibas, and Steve Y. Oudot. Manifold Reconstruction in Arbitrary Dimensions using Witness Complexes. *Discrete and Computational Geometry*, 42(1):37–70, 2009. 4

[8] Sergio Cabello, José Miguel Díaz-Báñez, Stefan Langerman, Carlos Seara, and Inma Ventura. Facility Location Problems in the Plane Based on Reverse Nearest Neighbor Queries. *European Journal of Operational Research*, 2009. 4

[9] Kenneth L. Clarkson. Nearest neighbor searching in metric spaces: Experimental results for $sb(s)$. Preliminary version presented at ALENEX99, 2003. URL `http://www.almaden.ibm.com/u/kclarkson/Msb/readme.html`. 4

[10] Kenneth L. Clarkson. Nearest-neighbor searching and metric space dimensions. In Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk, editors, *Nearest-Neighbor Methods for Learning and Vision: Theory and Practice*, pages 15–59. MIT Press, 2006. ISBN 0-262-19547-X. 3

[11] Kenneth L. Clarkson. An Algorithm for Approximate Closest-point Queries. *Proceedings of the Tenth Annual Symposium on Computational Geometry*, pages 160–164, 1994. 3

[12] Kenneth L. Clarkson. Nearest neighbor queries in metric spaces. *Discrete and Computational Geometry*, 22:63–93, 1999. 3

[13] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. *SCG '04: Proceedings of the Twentieth Annual Symposium on Computational Geometry*, pages 253–262, 2004. doi: http://doi.acm.org/10.1145/997817.997857. 3, 6

[14] Karina Figueroa and Rodrigo Paredes. Approximate direct and reverse nearest neighbor queries, and the k-nearest neighbor graph. *Proceedings of the 2nd International Workshop on Similarity Search and Applications (SISAP 2009)*, 2009. 4

[15] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity Search in High Dimensions via Hashing. *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529, 1999. 6, 20

[16] Leonidas J. Guibas and Steve Y. Oudot. Reconstruction using Witness Complexes. *Discrete and Computational Geometry*, 40(3):325–356, 2009. 4

[17] Sariel Har-Peled. A Replacement for Voronoi Diagrams of Near Linear Size. *Annual Symposium on Foundations of Computer Science*, 42:94–105, 2001. URL `http://valis.cs.uiuc.edu/~sariel/research/papers/01/avoronoi/avoronoi.pdf`. 1, 2, 3, 6, 12, 13

[18] Sariel Har-Peled. Approximation Algorithms in Geometry, 2009. URL `http://valis.cs.uiuc.edu/~sariel/teach/notes/aprx/`. Class Notes. 12, 13

[19] Piotr Indyk. Nearest Neighbors in High-dimensional Spaces. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 877–892. CRC Press, 2004. 3

[20] Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 604–613, 1998. 1, 2, 3, 6, 20

[21] James M. Kang, Mohamed F. Mokbel, Shashi Shekhar, Tian Xia, and Donghui Zhang. Continuous Evaluation of Monochromatic and Bichromatic Reverse Nearest Neighbors. *Proceedings of the IEEE 23rd International Conference on Data Engineering*, 2007. 4

[22] Jon M. Kleinberg. Two Algorithms for Nearest-Neighbor Search in High Dimensions. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 599–608, 1997. 3

[23] Flip Korn and S. Muthukrishnan. Influence Sets Based on Reverse Nearest Neighbor Queries. *ACM SIGMOD Record*, 29(2):201–212, 2000. 4

[24] Yokesh Kumar, Ravi Janardan, and Prosenjit Gupta. Efficient Algorithms for Reverse Proximity Query Problems. *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2008. 4

[25] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient Search for Approximate Nearest Neighbor in High Dimensional Spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000. 3

[26] Anil Maheshwari, Jan Vahrenhold, and Norbert Zeh. On Reverse Nearest Neighbor Queries. *Proceedings of Canadian Conference on Computational Geometry*, pages 128–132, 2002. 4

[27] Florian Pfender and Günter M. Ziegler. Kissing Numbers, Sphere Packings, and Some Unexpected Proofs. *Notices-American Mathematical Society*, 51:873–883, 2004. 4

[28] Yogish Sabharwal, Nishant Sharma, and Sandeep Sen. Nearest Neighbors Search using Point Location in Balls with Applications to Approximate Voronoi Decompositions. *Journal of Computer and System Sciences*, 72(6):955–977, 2006. 3, 6

[29] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-neighbor Methods in Learning and Vision: Theory and Practice*. MIT Press, 2005. 3

[30] Amit Singh, Hakan Ferhatosmanoglu, and Ali Şaman Tosun. High Dimensional Reverse Nearest Neighbor Queries. *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, pages 91–98, 2003. 4

[31] Ioana Stanoi, Divyakant Agrawal, and Amr El Abbadi. Reverse Nearest Neighbor Queries for Dynamic Databases. *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 44–53, 2000. 4

[32] Yufei Tao, Dimitris Papadias, and Xiang Lian. Reverse k-NN Search in Arbitrary Dimensionality. *Proceedings of the Thirtieth international Conference on Very Large Databases*, 30:744–755, 2004. 4

[33] Yufei Tao, Man Lung Yiu, and Nikos Mamoulis. Reverse Nearest Neighbor Search in Metric Spaces. *IEEE Transactions on Knowledge and Data Engineering*, pages 1239–1252, 2006. 4

Appendix A — The $\mathcal{A}(r, \varepsilon)$ data structure

In this section we show that a variant of the LSH data structure can solve the all-(r, ε)-$\mathcal{NN}$ problem. Recall from Section 2.3 that, given a locality-sensitive family of hash functions, the algorithm of [15, 20] builds a set of hash tables in which it hashes the data points, keeping only one point per bucket. For the all-(r, ε)-$\mathcal{NN}$ problem, we keep all the colliding entries in each bucket. This enables us to detect all the r-neighbors of the query point during the query phase. However, it also generates false positives, whose number might be so large as to make the procedure no more effective than a naive exhaustive search, should all these points be considered. This is why we maintain a false positive count, in order to quit if too many false positives are found. We also need to boost the sensitivity of our family of hash functions, in order to improve the chances of success of our online query procedure. Given a metric space $(X.\mathrm{d})$, a set U of integers and an (r_1, r_2, p_1, p_2)-sensitive hash family $\mathcal{F}$ (where p_1 and p_2 are constants less than 1) and an integer k, we define a new hash function $g : X \to U^k$ by letting $g(p) = (f_1(p), f_2(p), \ldots, f_k(p))$ where $p \in X$ and each f_i is drawn independently at random from the hash family $\mathcal{F}$. Call the family of such hash functions $\mathcal{G}(k)$. Since the hash key is now a k-dimensional vector, two keys $g_1(p)$ and $g_2(p)$ are equal if and only if all their coordinates match up, i.e. $g_1(p)_i = g_2(p)_i$ for all $i \in \{1, 2, \cdots, k\}$. The rest of the construction follows [15, 20], and we refer the reader to Algorithms 4 and 5 for the details.

Correctness of the output. Let us first assume that the query procedure terminates and prove that its output is correct with high probability. Our first task is to show that the number of false positives generated within an iteration of the main loop of Algorithm 5 is not too large with a constant probability:

Lemma A.1. *Fix an i w.r.t. Algorithm 5. The probability that there are more than 4τ false positives (i.e. points $p \in P$ that collide with q whereas $\mathrm{d}(q, p) > (1 + \varepsilon)r$) in the i-th loop of Algorithm 5 is at most $\frac{1}{4}$.*

Input : A metric space (X, d), a point set $P \subseteq X$, two real numbers $r, \varepsilon \geq 0$, an integer c, and an $(r, (1+\varepsilon)r, p_1, p_2)$-sensitive LSH family $\mathcal{F}$
Output: The $\mathcal{A}(r, \varepsilon)$ data structure

Let $\rho = \frac{\ln p_1}{\ln p_2}$, $\tau = 2n^{\rho}$ and $k = \frac{\log n}{\log 1/p_2}$
Create the k-dimensional hash family $\mathcal{G}$
for $i = 1$ *to* $c \log n$ **do**
  pick τ functions $\mathcal{G}^i = \{g_1, \ldots, g_\tau\}$ independently at random from $\mathcal{G}$
  Create τ hash tables $\mathcal{H}^i = \{H_1, \ldots, H_\tau\}$
  forall $p \in P$ **do**
    for $j = 1$ *to* τ **do**
      Insert p into H_j using the key $g_j(p)$
    end
  end
  Construct the data structure $\mathcal{A}_i(r, \varepsilon) = \mathcal{G}^i \sqcup \mathcal{H}^i$
end
Output $\mathcal{A}(r, \varepsilon) = \bigcup_i \mathcal{A}_i(r, \varepsilon)$

Algorithm 4: The pre-computation phase for all-(r, ε)-$\mathcal{NN}$, which builds the $\mathcal{A}(r, \varepsilon)$ data structure

Input : metric space (X, d), $\mathcal{A}(r, \varepsilon)$ data structure, query point $q \in X$

Let ρ, τ and k be defined as in Algorithm 4
for $i = 1$ *to* $c \log n$ **do**
  Initialize the output set: $S_i = \emptyset$
  Initialize false positives count: $pos = 0$
  Let $\mathcal{G}^i = \{g_1, \ldots, g_\tau\}$ be the functions and $\mathcal{H}^i = \{H_1, \ldots, H_\tau\}$ be the hash tables corresponding to $\mathcal{A}_i(r, \varepsilon)$
  for $j = 1$ *to* τ **do**
    Compute $g_j(q)$ and record the set of points C_j corresponding to the colliding entries in H_j
    forall $p \in C_j$ **do**
      if $\mathrm{d}(q, p) \leq (1+\varepsilon)r$ **then**
        Update the output set: $S_i = S_i \cup \{p\}$
      else
        Increase the false positives count: $pos = pos + 1$
        if $pos > 4\tau$ **then**
          Set $S_i = \emptyset$ and jump to end of line 18
        end
      end
    end
  end
end
Return $S = \bigcup_{i=1}^{c \log n} S_i$

Algorithm 5: The online query phase for all-(r, ε)-$\mathcal{NN}$

Proof. Take some integer $j \in \{1, 2, \cdots, \tau\}$. Recall that the hash family $\mathcal{G}$ is constructed in Algorithm 4 by concatenating $k = \frac{\log n}{\log 1/p_2}$ functions drawn from a $(r, (1+\varepsilon)r, p_1, p_2)$-sensitive family $\mathcal{F}$. Therefore, the probability that a given point $p \in P$ such that $\mathrm{d}(p,q) > r(1+\varepsilon)$ collides with q in the j-th hash table H_j is at most $p_2^k = p_2^{\frac{\ln n}{\ln 1/p_2}} = e^{\ln(p_2)\cdot\frac{\ln n}{\ln 1/p_2}} = \frac{1}{n}$. It follows that the expected number of points $p \in P$ with $\mathrm{d}(p,q) > r(1+\varepsilon)$ that collide with q in H_j is at most 1, and therefore that the expected number of such collisions in all the hash tables $H_1, \ldots, H_\tau$ is at most τ. By Markov's inequality, the probability that more than 4τ such collisions occur at iteration i of the algorithm is then at most $1/4$. □

Note that the test on line 9 of Algorithm 5 guarantees that the output S is always a subset of $\mathcal{NN}_P(q, r, \varepsilon)$. We will now use Lemma A.1 to show that S contains all the points of $\mathcal{NN}_P(q, r, 0)$ with high probability:

Lemma A.2. *Given a point $p \in \mathcal{NN}_P(q, r, 0)$, Algorithm 5 finds and outputs this point with high probability, i.e. $p \in S$ with high probability.*

Proof. Let us first consider a single loop of Algorithm 5 (i.e., fix an i w.r.t. Algorithm 5) and show that $p \in S_i$ with probability $\geq \frac{3}{5}$. This probability is clearly the same as the probability that there exists a function $g_j(\cdot)$ that hashes q and p to the same location ($g_j(q) = g_j(p)$). Since $d(q,p) \leq r$, the probability of a collision for a fixed j is at least $p_1^k = p_1^{\frac{\ln n}{\ln 1/p_2}} = n^{-\frac{\ln 1/p_1}{\ln 1/p_2}} = n^{-\rho}$. Therefore, the probability that no hash function g_j generates a collision is at most $(1-n^{-\rho})^\tau$, since τ functions are picked from $\mathcal{G}$ in each loop of Algorithm 5. Thus, the probability that one loop i of Algorithm 5 adds p to the set S_i is at least $1-(1-n^{-\rho})^\tau = 1-(1-n^{-\rho})^{2n^\rho} \geq 1-\frac{1}{e^2} > \frac{4}{5}$, where we substituted the value of $\tau = 2n^\rho$. The last thing to note is that for p to be actually output at the end of the i-th iteration, we must have no more than 4τ false positives, the probability of which is given by Lemma A.1. Putting these bounds together, we conclude that the probability of having $p \in S_i$ at the end of the i-th iteration is at least $\frac{4}{5} \times \frac{3}{4} = \frac{3}{5}$.

Now, the probability that p belongs to the output S is $1 - \mathrm{Prob}\left[p \notin S_i \ \forall i\right]$. Since there are $c \log n$ iterations in total, we deduce that the probability that $p \in S$ is at least $1 - \left(\frac{2}{5}\right)^{c \log n}$. □

The high probability statement of Theorem 2.5 is obtained by applying the union bound on the set $\mathcal{NN}_P(q, r, 0)$. This requires an appropriate choice of the constant c, but it is easy to see that this integer need not be too large. It also requires to solve the query by brute-force search when n is small.

Note that our analysis holds in fact for any given $n \geq |P|$: indeed, the probability that the number of false positives for P is larger than $4\tau = 8n^\rho$ can only decrease with n, while the probability of finding an approximate nearest neighbor remains the same since the number of trials is sufficient for the locality-sensitive hash function.

Query time and memory usage.

Lemma A.3. *Given a metric space (X, d), a set $P \subseteq X$ with n points, two parameters $r, \varepsilon \geq 0$, an integer c and a $(r, (1+\varepsilon)r, p_1, p_2)$-sensitive hash family $\mathcal{F}$ such that $\frac{\ln 1/p_1}{\ln 1/p_2} \leq \frac{1}{1+\varepsilon}$, the $\mathcal{A}(r, \varepsilon)$ data structure uses $O\left(n + n^{1+\frac{1}{1+\varepsilon}} \log n\right)$ space and is queried in $O\left(n^{\frac{1}{1+\varepsilon}} \log n + |\mathcal{NN}_P(q, r, \varepsilon)| \log n\right)$ time.*

Proof. Recall that we defined

$$\rho = \frac{\ln 1/p_1}{\ln 1/p_2} \leq \frac{1}{1+\varepsilon}.$$

At each iteration of the main loop of Algorithm 5, we perform τ hash probes for the query point q, and the cost of each probe is proportional to $k = \frac{\log n}{\log 1/p_2} = O(\log n)$ (recall from the beginning of Appendix A that p_2 is a constant less than 1). For each collision we perform a distance evaluation, which has a unit cost. The collisions involving points of $\mathcal{NN}_P(q, r, \varepsilon)$ have a total cumulative cost of $O(|\mathcal{NN}_P(q, r, \varepsilon)|)$; the collisions involving false positives have a total cumulative cost of $O(\tau) = O(n^\rho)$ since we allow a maximum of 4τ false positives. Putting these bounds together, we obtain the following time for a single iteration of the main loop: $O(n^\rho + |\mathcal{NN}_P(q, r, \varepsilon)|) = O(n^{1/(1+\varepsilon)} + |\mathcal{NN}_P(q, r, \varepsilon)|)$. The total query time is just this value scaled by $c \log n$.

The space bound is easily obtained by observing that storing the points requires $O(n)$ space (following the literature, we hide the d factor in the big-O notation), and we store k bits for each point in τ hash tables, resulting in a total storage of $O(n + n^{1+1/(1+\varepsilon)} \log n)$. □

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 Orsay Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399