Approximate Nearest Neighbors: Point Location in Equal Balls and Ring-Cover Trees

Lukas Arnold

April 20, 2018

1 Introduction

Nearest neighbor search (NNS) deals with the problem of finding a point p of a set $P = \{p_1, \ldots, p_n\}$ closest to a query point q within a metric space X, usually $\mathbf{R}^d$. Given distance functions $d(.,.)$, P should be preprocessed in a way so that the query can be handled most efficiently. The problem is of vital interest for a wide range of applications that require similarity searching in high-dimensional space, such as text retrieval, image and video databases, machine learning and pattern recognition, where the number of features can easily reach several thousands and are therefore affected by the *curse of dimensionality*: As the number of dimensions increases, computation time often increases polynomially or exponentially. NNS algorithms for high dimensions known previous to the paper *Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality* published by Piotr Indyk and Rajeev Motwani in 1998 [2] suffered from either of two phenomena: either preprocessing time was low, but query time was linear by the number of points n and dimensionality d, or query time was sublinear in n and polynomial in d, but had preprocessing costs rising exponentially with the number of dimensions d. In this manuscript, an algorithmic approach is presented that yields significant improvement of these conditions. Indyk and Motwani propose a solution in which this *curse of dimensionality* could be overcome by finding an approximate method for the NNS problem: Find a point p approximately closest to q, so that $\forall p' \in P : d(p, q) \leq (1+\epsilon)d(p', q)$. This is known as the **ϵ-approximate nearest neighbor** (ϵ-NNS) problem.

1.1 Overview

Two fundamental tools presented in the paper to reach this aim are discussed in the manuscript. One is, that the ϵ-NNS problem can be reduced to a new problem: *Point location in equal balls* (PLEB). The other is a data structure for efficient searching called *ring-cover trees*.

Indyk and Motwani present two algorithms to solve the problem: One for $0 < \epsilon < 1$ that uses $\tilde{O}(n) \times O(\frac{1}{\epsilon})^d$ preprocessing time and $\tilde{O}(d)$ query time. This is called the *bucketing method*. And one for $\epsilon > 0$ that uses $(n \cdot d)^{O(1)}$ preprocessing time and achieves the goal of having maximum $\tilde{O}(d)$ query time, called *locality-sensitive hashing*, that however will not be discussed as it is presented by other seminar contributors.

2 NNS to PLEB reduction

In order to derive the algorithms, the (approximate) nearest neighbor search problem is converted into a (approximate) *point location in equal balls* problem. By relaxing the *nearest* neighbor problem into a *near* neighbor problem, computational cost is traded for result precision.

An $\mathbf{R}^d$ space under some l_p-norm is assumed. A *ball* B centered at p with radius r is defined as $B(p, r) = \{q \in X | d(p, q) \leq r\}$.

The classical PLEB would be solved by a data structure that, for n balls $B(p, r)$ (for all $p \in P = \{p_1, \ldots, p_n\}$), returns p_i if $q \in B(p_i, r)$, or returns NO if no such point can be found.

The classical PLEB can be replaced by an ϵ-approximate PLEB problem, that is formulated as follows:

1. If there is a $p_i \in C$ with $q \in B(p_i, r)$ (small ball), return YES and a point p_i' with $q \in B(p_i', (1 + \epsilon)r$ (large ball).
2. If there is no $p_i \in C$ with $q \in B(p_i, (1 + \epsilon)r)$, return NO.
3. Otherwise, return either YES or NO.

See figure 1 for an illustration for the PLEB and ϵ-PLEB queries.

The reduction from ϵ-NNS to ϵ-PLEB makes use of a data structure called *ring-cover tree*, that will be presented in section 3. A simple reduction can be sketched as below. Please note, that a more complex but storage-wise cheaper implementation of this algorithm has also been presented by Indyk and Motwani.

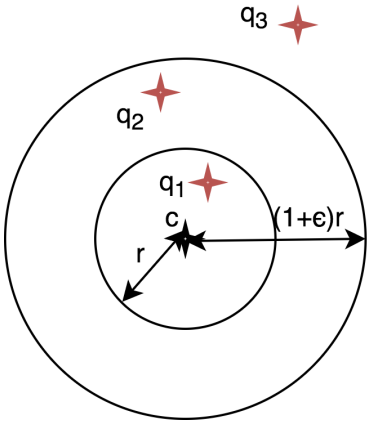

Figure 1: A small ball ($B(p,r)$) and large ball ($B(p,(1+\epsilon)r)$) are shown. If q is within the small ball (q_1), return YES and a point p_i' with $q \in B(p_i', (1+\epsilon)r$, e.g. q_2. If it is within the big ball, but not in the small ball (q_2), return either YES or NO. For the query point outside the balls (q_3), return NO. This structure is also called a *ring*.

1. For all $l \in \{(1+\epsilon)^0, (1+\epsilon)^1, \ldots, R\}$ where R is the ratio of the smallest to the largest distance of points in P, generate a sequence of balls $B^l = \{B(p_1, l), \ldots, B(p_n, l)\}$.
2. For query q, find the minimal l for which $\exists i : q \in B(p_i, l)$ via binary search.
3. Return p_i as an approximate nearest neighbor.

3 Ring-Cover Trees

A *ring-cover tree* is a data structure for PLEB and ϵ-PLEB that is based on *rings* and *covers*.

Definition 1 A *ring* $R(p, r_1, r_2)$ is an $(\alpha_1, \alpha_2, \beta)$-ring separator for P, if the number of points in $B(p, r_1) \geq \alpha_1 \cdot n$ and the number of points outside $B(p, r_2) \geq \alpha_2 \cdot n$ and $\frac{r_2}{r_1} = \beta$. Please refer to figure 1 for an example of an $\left(\frac{1}{3}, \frac{1}{3}, (1+\epsilon)\right)$-ring.

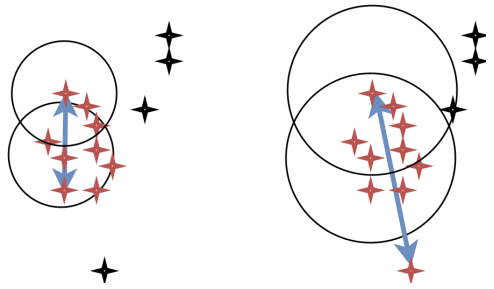

Figure 2: Two cases are shown. For both, the set S is formed of all the red points. The red and black points together form the point set P. The blue line is the diameter $\Delta(S)$. In the image left, for no point more than half of all points are inside the ball of radius $\Delta(S)$. It is therefore a (1,0.5)-cluster. For the image on the right, one red point was added, thus the ball radius $\Delta(S)$ becomes large. As more than half of the points are inside the ball, the set of red points is no longer a cluster.

Definition 2 A (γ, δ)-*cluster* is a set $S \subset P$ if $\forall p \in S$, the number of points in ball $B(p, \gamma\Delta(S)) \leq \delta \cdot n$, and $\Delta(S)$ being the diameter of S. An example of an (1,0.5)-cluster can be seen in figure 2.

Definition 3 A (b, c, d)-*cover* for $S \subset P$ is a sequence of sets $A_1, \ldots, A_l$ of sets $A_i \subset P$, if $\exists r \geq d\Delta(\cup_i A_i) : S \subset A \wedge \forall i \in \{1, \ldots, l\}$ such that the number of all points in the small balls $B(p, r) \leq b \cdot |A_i|$ and $|A_i| \leq c \cdot n$. An example of a $(2.5, \frac{5}{12}, 1)$-cover is shown in figure 3.

One of the essential properties of each point set P is, that it is either an (α, α, β)-*ring node* or a (b, α, d)-*cover node*. This allows us to construct a ring-cover tree for any data set. As a simpler example, the quadtree is taken: For a point set P, the space is divided into four equally-sized windows, and for each window that contains a point p, a tree node is added to the data structure. This is repeated recursively, until for every point p, a node exists. See figure 4 for an example. A ring-cover tree would not split the data set into equal windows, but into either rings or covers until a small amount of points has been reached, where the $PLEB$ base case can be applied. An illustration is given in figure 5.

Ring node P is a ring node if it has an (α, α, β)-ring separator $R(p, r, \beta r)$. In this case, its children are defined to be S_1 that is the set of points that

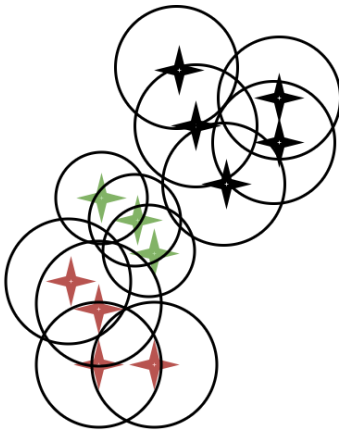

Figure 3: An example of a cover with three sets $A_i \subset P$: blue, green and black. For no set, more than $b = 2.5\times$ points are within the balls with radius $1 \cdot \Delta(A_i)$ of the subset, and no subset contains more than $c = \frac{5}{12}$ of the total points P.

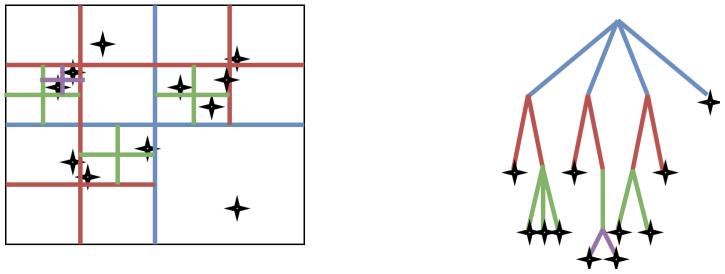

Figure 4: An example of a quadtree. The point set P consists of all the black points in the $2d$ space. It is split into four equal windows until there is only one point in each window. For each layer of splitting, the depth of quadtree increases. The image on the left shows the point set and the windows, the image on the right depicts the corresponding data tree.

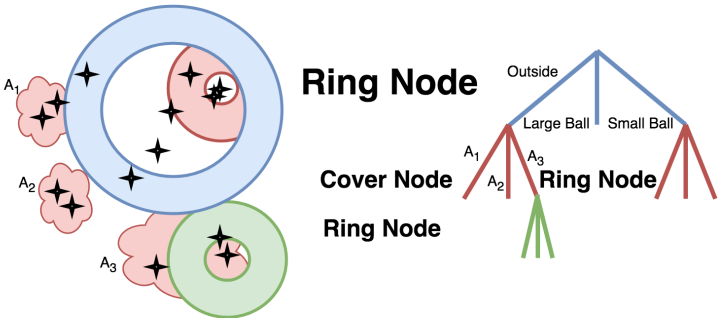

Figure 5: The point set P is split in either rings or covers, until a small amount of point (2 in this example) has been reached. The image on the left shows the data with the splittings by either rings or covers (cloud-shaped). The image on the right shows the corresponding tree.

include all points inside ball $B(p, \beta r)$ and S_2 that contains all points outside of ball $B(p, r)$. An efficient search procedure that returns the ϵ-approximate nearest neighbor is described below. For any P, the search can be restricted to one of its children by only a few tests. For $\text{SEARCH}(q, P)$, the procedure is :

1. if $q \in B(p, r(1 + \frac{1}{\epsilon}))$, return $\text{SEARCH}(q, S_1)$.
2. else, compute $p' = \text{SEARCH}(q, S_2)$ and return $\min_q(p, p')$.

Cover node If P is not a ring node, it is a cover node with a (b, α, d)-cover for some $S \subset P$, such that $|S| \geq (1 - 2\alpha)n \wedge d = \frac{1}{(2\beta+1)\log_b n}$. Then S_i is defined as $P \cap \cup_{p \in A_i} B(p, r)$ and $S_0 = S - A$. Then generate a PLEB with balls $B(p, r_i)$ for $p \in A$ for all $r_i = \frac{r_0}{(1+\epsilon)^i}$ with $i \in \{1, \ldots, k\}$ and $k = \log_{1+\epsilon} \frac{1+\frac{1}{\epsilon}\log_b n}{\gamma} + 1$. Then the procedure $\text{SEARCH}(q, P)$ is as follows:

1. if $q \notin B(a, r_0)$, compute for all $a \in A$ $p = \text{SEARCH}(q, P - A)$, and return $\min_q(p, a)$ for a deliberately chosen a.
2. otherwise, if $q \in B(A, r_0)$ for some element $a \in A$, but $q \notin B(a', r_k)$ for all $a' \in A$, then find a ϵ-nearest neighbor p of q in A via binary search, compute $p' = \text{SEARCH}(q, P - A)$ and return $\min_q(p, p')$.
3. else, return $\text{SEARCH}(q, S_i)$.

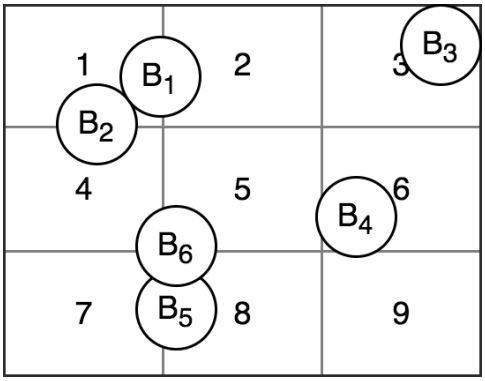

Figure 6: A grid containing balls $B_1, \ldots, B_6$ (not calculated, shown only as example). The corresponding hash table is shown in table 1.

Grid cell	1	2	3	4	5	6	7	8	9
Balls B_i within cell	B_1, B_2	B_1	B_3	B_2, B_6	B_4, B_6	B_4	B_5, B_6	B_5, B_6	B_4

Table 1: The hash table generated from the example in figure 6. To answer a query q, it is only necessary to compute the grid cell that contains q in order to return the corresponding value i.

4 Point Location in Equal Balls

Given the ring-cover tree, Indyk and Motwani have introduced two algorithms to solve the ϵ-approximate PLEB. The *bucketing method* and *locality-sensitive hashing*. As locality-sensitive hashing is covered by other talks, only the bucketing method is presented.

4.1 Bucketing method

The *bucketing algorithm* is based on the Elias bucketing algorithm [6]. A uniform grid is created with spacing $\epsilon \cdot d^{-1/p}$ for any l_p norm. This grid forms a cover for P. For each ball B_i, $\overline{B}_i$ is defined as the set of grid cells that intersect B_i. All elements from $\cup_i \overline{B}_i$ are stored in a hash table.

After the preprocessing, query q can be answered by computing the grid cell containing q and check the hash table if it is stored in the table. Using this method, $O(n) \times O(\frac{1}{\epsilon})^d$ preprocessing time and $O(1)$ evaluations per query can be achieved for $0 < \epsilon < 1$. An example of a grid is shown in figure 6, with the corresponding hash table 1.

5 Conclusion

The approximate NNS problem has been addressed by many throughout the years. Kushilevitz et al. have presented a solution with polylogarithmic query time and polynomial preprocessing time in 2000 [4], and Krauthgamer and Lee have presented a solution requiring logarithmic query time in 2002 [3]. The methods presented in the discussed paper, including locality-sensitive hashing, have grown to be one of the most popular algorithms, applied to a wide range of areas, including web clustering, biology, computer vision and computational linguistics [1], with ongoing research in terms of accuracy and efficiency [5].

References

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[2] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[3] R. Krauthgamer and J. R. Lee. Navigating nets: simple algorithms for proximity search. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 798–807. Society for Industrial and Applied Mathematics, 2004.

[4] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000.

[5] J. Wang, T. Zhang, N. Sebe, H. T. Shen, et al. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):769–790, 2018.

[6] T. A. Welch. Bounds on information retrieval efficiency in static file structures. Technical report, MIT, Cambridge MA, 1971.