On the Asymptotic Behavior of Nearest Neighbor Search Using Pivot-Based Indexes

Benjamin Bustos
PRISMA Research Group
Department of Computer Science
University of Chile
bebustos@dcc.uchile.cl

Nelson Morales
DELPHOS Lab
AMTC
University of Chile
nmorales@ing.uchile.cl

ABSTRACT

This paper presents an asymptotic analysis for the nearest neighbor search with pivot-based indexes. We extend a previous analysis based on range queries with fixed tolerance radius, because there is a probability that the nearest neighbor is missed. We introduce a probabilistic analysis and then we show the expected search cost for range-optimal algorithms. Finally, we also show the analysis of the proposed search algorithm taking into account the extra CPU time, which leads to further insights on the efficiency of different implementations of this algorithm.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content analysis and indexing—*indexing methods*

General Terms

Theory

Keywords

Nearest-neighbor search, pivot-based indexing, asymptotic analysis

1. INTRODUCTION

The concept of similarity search has applications in a vast number of fields. For example, content-based retrieval of similar objects in multimedia databases can be very useful for industrial applications, medicine, molecular biology, among others. Other applications for similarity search include machine learning and classification (where a new element must be classified according to its closest existing element); image quantization and compression (where only some vectors can be represented and those that cannot must be coded as their closest representable point); text retrieval (where we look for words in a text database allowing a small number of errors, or we look for documents which are similar to a given query or document); sequence comparison in computational biology (where we want to find a DNA or protein sequence in a database allowing some errors due to typical variations); etc.

All those applications have some common characteristics. There is a universe of *objects* and a *distance function* defined among them, which in many interesting cases satisfies the properties of a metric (strict positiveness, symmetry, and the triangle inequality). That is, the universe of objects together with the distance function form a *metric space*. The similarity between objects is given by their distance in the defined metric space: the smaller the distance between two objects, the more similar they are. Thus, for a given application there is a finite *dataset*, which is a subset of the universe of objects. Later, given a new object from the universe, one wants to retrieve similar elements found in the dataset.

An important similarity query type is the *nearest neighbor (NN) search*. In this type of query, one wants to retrieve from the dataset the most similar object to the query. Similarly, a k-NN search returns the k closest objects from the dataset to the query. Both queries can be easily done by a sequential scan over the dataset, computing the distance from the query to every object in the set, and returning the closest ones. However, this may be too expensive if the dataset is very large or if the distance function is expensive to compute. Thus, it is common to define the complexity of a similarity search in metric spaces as the number of distances computations required to answer the query.

For this reason, one usually preprocesses the dataset to build an index to avoid unnecessary distance computations at query time. For example, *pivot-based indexes* uses the distances between the objects from the dataset to some selected objects (the pivots) to discard objects from the similarity search without measuring the distance to the query (see Section 2.2 for details). In this way, the index can be used to return the nearest neighbor to the query, avoiding (hopefully) the sequential scan.

This paper presents a formal analysis of nearest neighbor search algorithms for pivot-based indexes. As noted by Navarro [13], asymptotic analysis is a very important tool that may lead us to better understand the behavior of index structures. We base our analysis on a range-optimal algorithm that works on pivot tables. Then, we extend our analysis by considering the extra CPU time (not expended in distance computations) required to perform the search, and we argue that this is necessary for making the analysis of NN search with pivot-based indexes meaningful.

Table 1: Notation used in this paper

$(\mathbb{U}, \delta)$	Metric space
$\mathbb{S} \subset \mathbb{U}$	Dataset
$n = \|\mathbb{S}\|$	Size of the dataset
$s, s_i \in \mathbb{S}$	An element of the dataset
$cost(\delta)$	Complexity of function $\delta(\cdot, \cdot)$
(q, r)	Range query with tolerance r
$\mathbb{P} \subset \mathbb{S}$	Set of pivots
$\theta \in P$	A pivot
$p = \|\mathbb{P}\|$	Number of selected pivots
$lb(q, s)$	Pivot-based lower bound distance
$\nu(r)$	Probability of not discarding an element in a (q, r) query
$f(x)$	Distance distribution of δ
$F(x) = \int_{-\infty}^{x} f(t)dt$	Cumulative distribution of $f(x)$
δ^*	Distance to the NN
$f^*(x)$	Distribution of δ^*
$F^*(x) = \int_{-\infty}^{x} f^*(t)dt$	Cumulative distribution of $f^*(x)$
$\pi = Pr(\delta^* > \mathbb{E}(\delta^*))$	Probability that δ^* is larger than its expected value

The contributions of this paper are:

- We improve a previous analysis of nearest neighbor search algorithms, to account the probability of failing to find the nearest neighbor.
- We present a framework for probabilistic analysis of nearest neighbor search, and we apply it to the case of uniform distance distribution.
- We present a nearest neighbor algorithm for pivot-based indexes, and we analyze it and prove that it is range-optimal.
- We take into account the extra CPU time in our analysis, that may be involved in the nearest neighbor search.

This paper is organized as follows. Section 2 presents the basics concepts of similarity search in metric spaces and explains the pivot-based indexing method. Section 3 present an analysis of pivot-based NN search algorithms that uses a fixed-search radius. Section 4 presents an introduction to probabilistic analysis for pivot-based NN search algorithms. Section 5 presents a range-optimal algorithms for pivot-based indexes and its analysis. Section 6 shows a model cost that includes the extra CPU time. Finally, Section 7 concludes the paper.

2. BASIC CONCEPTS

This section introduces the basic concepts on similarity search in metric spaces and pivot-based indexing. Table 1 shows the notation used in this paper.

2.1 Similarity search in metric spaces

Let $\mathbb{U}$ be the "universe" of valid objects, and let $\delta : \mathbb{U} \times \mathbb{U} \to \mathbb{R}^+$ be a metric distance function (i.e., it holds the properties of strict positiveness, symmetry, and the triangle inequality). The pair $(\mathbb{U}, \delta)$ is a metric space.

Let $\mathbb{S} \subset \mathbb{U}$ be a data collection (or dataset), and let $q \in \mathbb{U}$ be a query object. There are two typical similarity queries in metric spaces:

- *Range query.* Given a search radius $r \in \mathbb{R}^+$, the range query (q, r) reports all objects from the dataset that are within a distance r to q, that is, $(q, r) = \{s \in \mathbb{S}, \delta(s, q) \leq r\}$. The subspace $\mathbb{V} \subset \mathbb{U}$ defined by q and r (i.e., $\forall v \in \mathbb{V}\ \delta(v, q) \leq r$ and $\forall x \in \mathbb{X} - \mathbb{V}\ \delta(x, q) > r$) is called the *query ball*.
- *k nearest neighbors query (k-NN).* It reports the k closest objects from $\mathbb{S}$ to q. That is, it returns a set $\mathbb{C} \subseteq \mathbb{S}$ such that $|\mathbb{S}| = k$ and $\forall x \in \mathbb{C}, y \in \mathbb{S} - \mathbb{C}, \delta(x, q) \leq \delta(y, q)$. In this paper, we are interested in the case $k = 1$, which is known as the *nearest-neighbor* (NN) search.

Both types of similarity queries can be solved by performing a sequential scan of the dataset. However, this may be too slow for practical applications, where the dataset may contain millions of objects. Thus, efficient similarity search algorithms works on top of index structures (known as *metric access methods*), that try to discard objects from the dataset without computing their distance to q, thus avoiding the sequential scan. While there are several indexing techniques for metric spaces, in this paper we are specially interested in *pivot-based indexing techniques*.

2.2 Pivot-based indexing

Pivot-based indexes select a number of *pivot* objects from $\mathbb{S}$, and classify all the other objects according to their distance from the pivots. The canonical pivot-based range query algorithm works as follows: Given a range query (q, r) and a set of p pivots $\{\theta_1, \ldots, \theta_p\}, \theta_i \in \mathbb{S}$, by the triangle inequality it follows for any $u \in \mathbb{U}$ and $1 \leq i \leq p$ that $\delta(q, u) \geq |\delta(\theta_i, u) - \delta(\theta_i, q)|$. The objects $s \in \mathbb{S}$ of interest are those that satisfy $\delta(q, s) \leq r$, so one can exclude all the objects that satisfy $|\delta(\theta_i, s) - \delta(\theta_i, q)| > r$ for some pivot θ_i, without actually evaluating $\delta(q, s)$. This discarding criterion is known as the *pivot exclusion condition*.

The pivot-based index consists of the pn precomputed distances $\delta(\theta_i, s)$ between every pivot and every object of the data collection. At query time, the search algorithm computes the p distances between the pivots and q, $\delta(\theta_i, q)$. Those distance calculations correspond the *internal complexity* of the algorithm, and this complexity is fixed if there is a fixed number of pivots. Then, it tries to discard objects by applying the exclusion condition. The list of objects $\{s_1, \ldots, s_m\} \subseteq \mathbb{S}$ that cannot be discarded, known as the *object candidate list*, must be checked directly against the query. These additional distance calculations $\delta(s_i, q)$ correspond to the *external complexity* of the algorithm. The total complexity of the search is the sum of the internal and the external complexities.

Examples of pivot-based indexes are *Burkhard-Keller Tree* [5], *Fixed-Queries Tree* (FQT) [1], *Fixed-Height FQT* [1], *Fixed Queries Array* [8], *Vantage Point Tree* [16], *Multi Vantage Point Tree* [4], *Excluded Middle Vantage Point Forest* [17], *AESA* [15], *Linear AESA* [12], *Spaghettis* [7], *Maximum Pruning* [14], and *Dynamic SSS* [6].

NN search algorithms with pivot-based indexes have usually been built over the range query algorithm. These techniques include [9]:

- *Increasing radius*: the search performs several range queries with increasing value of r, until at least one object lies inside the ball (q, r). If more than one object is found, the closest one to q is the NN.
- *Backtracking with decreasing radius*: the search starts with $r = \infty$, and it reduces r every time it computes a distance $\delta(q, s)$ that is smaller than r, keeping the closest object to q found so far. The search finishes when all objects have been checked.

2.3 Standard cost model

The total time for answering similarity queries in metric spaces is defined as [9]

$$\#\ dist.\ comp. \times cost(\delta) + \ extra\ CPU\ time\ +\ \mathrm{I}/O\ time,$$

where $cost(\delta)$ is the complexity of function $\delta(\cdot, \cdot)$. It has been argued [9, 18] that the most important cost in a similarity search is the distance computations, thus the cost model is usually simplified to

$$\#\ dist.\ comp.$$

See Navarro [13] for a survey on formal analyses of indexing techniques and algorithms for similarity search in metric spaces.

3. ANALYSIS OF NN SEARCH USING FIXED-RANGE QUERIES

In this section, we revisit the analysis of NN search algorithms based on range queries with fixed radius, and we extend it. Let $\delta^*(q)$ be the distance to the nearest neighbor of q (from now on, let us write δ^* for short). A common way of studying the cost of search algorithms for 1-NN, is to take into account that if δ^* was known, then 1-NN reduces to the range query (q, δ^*), therefore the expected cost of a 1-NN search is that of a range query $(q, \mathbb{E}(\delta^*))$ [2]. However, such analysis does not consider that the cost of a query search is not linear in the radius, hence the expected cost of using a query search with radius δ^* is not the same as the cost of a query search with radius $\mathbb{E}(\delta^*)$. It also does not consider that searching with fixed radius $\mathbb{E}(\delta^*)$ may not find the nearest neighbor.

3.1 Range based Nearest-Neighbor Queries

Given a range-optimal algorithm (see Section 5) for finding the NN in multidimensional indexes, Bï£¡hm [2] suggests reducing its analysis to the analysis of a range query using a tolerance radius for the range query equal to the expected value $\mathbb{E}(\delta^*)$ of the distance to the nearest neighbor. Concretely, following the analysis by Navarro [13], we have that if the distances between the elements of the database and from these to the query are independent and identically distributed, and if we call T_R the cost of such an algorithm, we have that

$$\mathbb{E}(T_R) = p + n\,\nu(\mathbb{E}(\delta^*))^p, \tag{1}$$

where p is the number of pivots and $\nu(\cdot)$ is the probability of not discarding an object using the p pivots.

While this analysis may produce a good approximation of the expected cost of the algorithm, we observe that it does not consider the fact that any NN search algorithm resorting to use a range query with fixed value of r_0 will fail (return no elements) with probability $Pr(\delta^* > r_0)$. In particular, the procedure above will require to perform at least one more distance calculation with probability $\pi = Pr(\delta^* > \mathbb{E}(\delta^*))$.

As using a fixed radius r_0 (like $\mathbb{E}(\delta^*)$) may produce no results, the only way to guarantee that the nearest neighbor will be found is to update the query radius. Some of the possibilities that we visualize are:

- Doing a linear search in the case of failure. This has expected cost $p + n\big(\nu(\mathbb{E}(\delta^*)^p)\big) + \pi n$, therefore being of order n.
- Multiplying r_0 by some factor greater than one, do the search and continue updating or stopping depending on the results of the search.
- Updating r by setting $r = \mathbb{E}\big(\delta^* | \delta^* > \mathbb{E}(\delta^*)\big)$, and update consequently depending on the search result.

Note that any of these schemes (and any algorithm performing a range query with radius $\mathbb{E}(\delta)$) will have an expected cost that is strictly larger than the one from Eq. (1).

Notice also that, in particular, when the distances are not bounded (i.e., the distance distribution has an infinite support set), we have that any search algorithm using a finite query radius r_0 is not exact, as it will fail with probability $Pr(\delta^* > r_0)$.

4. PROBABILISTIC ANALYSIS

This section aims to present a brief introduction to the probabilistic analysis that would be required in order to properly understand the cost of NN algorithms for pivot-based indexes. We present some basic calculations and then apply them to the specific case of the uniform distribution.

4.1 Distribution and expected values of δ^*

We need first to calculate the probability density of δ^*, that is the distance to the nearest neighbor, when the pivots are chosen randomly and independently.

As the pivots are chosen randomly, the distance distribution to any pivot is $f(x)$ (the distance distribution to the objects in the database), and since $\delta^*(q) = \min_{i=1}^{n} \delta(q, s_i)$, we have that $\delta^*(q) \geq z \iff \delta(q, s_i) \geq z$ for any $i = 1, \ldots, n$. Using that $Pr(\delta(q, s_i) \geq z) = 1 - F(z)$, and independence, it follows that $Pr\big(\delta^* \geq z\big) = (1 - F(z))^n$. Therefore, $Pr(\delta^* \leq z) = F^*(z) = 1 - (1 - F(z))^n$. That is, $\delta^* \sim f^*(z)$ with

$$f^*(z) = n f(z)(1 - F(z))^{n-1}, \tag{2}$$

and

$$\mathbb{E}\big(\delta^*\big) \ = \ \int_{\infty}^{s} z n f(z)(1 - F(z))^{n-1} dz. \tag{3}$$

Notice that in the general case of k-NN, we have that

$$Pr\big(\delta^*(x) \geq z\big) = \binom{n}{k} F(z)^{k-1}(1 - F(z))^{n-k+1}.$$

Finally, recall that the probability of not discarding an element when performing a search with radius r is (see Navarro [13]):

$$\nu(r) = \int_0^1 f(s)\,[F(s+r) - F(s-r)]\,ds.$$

4.2 Uniform distribution case

When the distance δ follows a uniform distribution in $[0,1]$, $f(x) = 1, F(x) = x$ (within [0,1]) and therefore

$$f^*(z) = n(1-z)^{n-1},\ F^*(z) = 1 - (1-z)^n$$

and, integrating by parts

$$\mathbb{E}(\delta^*) \quad = \quad \int_0^1 zn(1-z)^{n-1}dz = \frac{1}{n+1}.$$

Also $\pi = Pr\left(\delta^* > \frac{1}{n+1}\right) = \left(\frac{n}{n+1}\right)^n \rightarrow \frac{1}{e}$ as n grows. This means that in about 1/3 of the opportunities that the algorithm is used, it will fail to find the nearest neighbor and therefore at least one additional range query will be needed.

We also obtain that $\nu(r) = r(2-r)$.

As $\nu(\cdot)$ is a concave positive function and $p \geq 1$, we have that $\mathbb{E}(\nu(\delta^*)) \leq \nu(\mathbb{E}(\delta^*))$. It follows that as an algorithm using $\mathbb{E}(\delta^*)$ as a search radius will have an expected cost $\mathbb{E}(T_R) \geq p + n\nu\left(\mathbb{E}(\delta^*)\right)^p$. We conclude that

$$\mathbb{E}(T_R) \geq p + n\nu\left(\mathbb{E}(\delta^*)\right)^p \geq p + n\mathbb{E}\left(\nu(\delta^*)\right)^p \qquad (4)$$

The right side of Eq. (4) is actually the expected cost of an algorithm that *knows* δ^* and performs one range query with that radius, which is the definition of a range-optimal search algorithm.

5. RANGE-OPTIMAL ALGORITHMS

A NN search algorithm is *range-optimal* if it computes the same number of distances than the canonical range query algorithm using $r = \delta^*$ as search radius. This is an interesting property, because it means that for such an algorithm there is no inherent advantage of having or estimating δ^* a priori (without knowing the object that is actually the NN) to perform the search. An example of a range-optimal algorithm is the incremental search algorithm by Hjaltason and Samet [11], that works on hierarchical index structures. Berchtold et al. [3] proved that this algorithm is range-optimal.

The algorithms for NN search mentioned in Section 2.2 and Section 3 are not range-optimal. So, we now describe a NN search algorithm that uses the pivot-based index, and we prove that this algorithm is range-optimal.

The algorithm starts by computing the distances between all pivots and the query object q. The pivot whose distance to q is minimum (*mindist*) is the first NN candidate. Then, the algorithm computes the lower-bound distances from q to all objects $s \in \mathbb{S}$ (excluding the pivots). A lower-bound distance is computed as $lb(q,s) = \max_{i=1}^{p} |\delta(\theta_i, s) - \delta(\theta_i, q)|$. Next, the algorithm sorts the objects $s \in \mathbb{S}$ in ascending order according to their lower-bound distances to q. Starting with the the object u with smallest lower bound distance, the algorithm applies the pivot exclusion criterion using as tolerance radius the distance from the candidate NN to the query object. If s cannot be discarded, it computes the distance between s and q. If this distance is smaller than *mindist*, it sets u as the new NN candidate and updates *mindist*. The process ends when all the objects from $\mathbb{S}$ have been checked or if the lower bound distance of the next object in the list is greater than *mindist* (i.e., no other object can be closer to q than the actual NN candidate). Algorithm 1 shows the pseudocode for this algorithm.

Algorithm 1 Range-optimal pivot-based NN search algorithm

Require: $q \in \mathbb{U}$
Ensure: NN is the nearest neighbor of q in $\mathbb{S}$
1: $mindist \leftarrow \min_{i=1}^{p}\{\delta(\theta_i, q)\}$
2: $NN \leftarrow \theta_{\arg\min_{i=1}^{p}\{\delta(\theta_i,q)\}}$
{Computing lower bound distances}
3: **for all** $s_i \in \mathbb{S} - \mathbb{P}$ **do**
4: $lb(q, s_i) \leftarrow 0$
5: **for all** $\theta_j \in \mathbb{P}$ **do**
6: **if** $|\delta(s_i, \theta_j) - \delta(\theta_j, q)| > lb(q, s_i)$ **then**
7: $lb(q, s_i) \leftarrow |\delta(s_i, \theta_j) - \delta(\theta_j, q)|$
8: **end if**
9: **end for**
10: **end for**
{Sorting objects in $\mathbb{S} - \mathbb{P}$ by ascending lower bound}
11: $\mathbb{S}' \leftarrow Sort(\mathbb{S} - \mathbb{P})$
{Searching for NN}
12: **for** $i = 1$ to $n - p$ **do**
13: **if** $lb(q, s'_i) > mindist$ **then**
14: break
15: **end if**
16: **if** $\delta(q, s'_i) < mindist$ **then**
17: $mindist \leftarrow \delta(q, s'_i)$
18: $NN \leftarrow s'_i$
19: **end if**
20: **end for**
21: **return** NN

We now prove that Algorithm 1 is range-optimal.

LEMMA 5.1. *Algorithm 1 returns the correct NN.*

PROOF. By contradiction, suppose that for a query object q the algorithm returns an object $s \in \mathbb{S}$ such that $\delta(q,s) > \delta^*$, and let s^* be the real NN. This means that the algorithm stopped checking objects (lines 13 and 14 of the pseudocode) before finding s^*. Thus, $lb(q, s^*) > \delta^*$ because the algorithm checks the objects in ascending lower bound distance and the condition on line 13 had to be true before reaching s^*. But $lb(q, s^*) \leq \delta^*$, because $lb(q, s^*)$ is a lower bound distance of δ, which leads to the contradiction. □

LEMMA 5.2. *Algorithm 1 does not compute any distance from q to an object $s \in \mathbb{S}$ such that $lb(q,s) > \delta^*(q)$*

PROOF. The algorithm sorts the objects $s \in \mathbb{S} - \mathbb{P}$ by lower bound distance (line 11 of the pseudocode), it checks the objects in that order, and it stops the search as soon as $lb(q,s) > mindist = \delta^*$ (by Lemma 5.1). Thus, the algorithm only computes distances for those s such that $lb(q,s) \leq \delta^*$. □

THEOREM 5.3. *Algorithm 1 is range-optimal.*

PROOF. It follows directly from Lemmas 5.1 and 5.2. □

Notice that because a range-optimal search algorithm is equivalent to one doing a range search with radius $r = \delta^*$, we obtain that the cost of this algorithm is $T = p + n\nu(\delta^*)^p$ (with distribution f^* of δ^*), and its expected cost is the one given by Eq. (4). Therefore, if $\nu(\cdot)$ is concave (as we showed for uniform distance distribution), Algorithm 1 is more efficient than any possible strategy that uses a fixed-search radius for finding the nearest neighbor.

6. EXTENDED MODEL COST

In the last section, we showed that range-optimal algorithms computes the minimum number of distances (given a fixed set of pivots) needed to find the correct nearest neighbor. Thus, under the standard cost model for searching in metric spaces, any range-optimal algorithm is equivalent. However, we will show that this is far from true, and that it is important to consider the extra CPU time to obtain further insights from the theoretical analysis.

6.1 A CPU-expensive range-optimal NN algorithm

The algorithm we devise is very simple: We will determine the value of δ^* so we can perform a range query with radius $r = \delta^*$ and retrieve the nearest neighbor. The trick in the algorithm is that we will determine δ^* at cost zero and therefore the expected cost of such an algorithm will be the same of Eq. (4):

$$\mathbb{E}(T_R) = p + n\mathbb{E}\left(\nu(\delta^*)^p\right).$$

To determine δ^*, we use the following procedure. Let q be the query object. We start evaluating the distances $d_i = \delta(q, \theta_i)$ for $i = 1, \ldots, p$ and constructing the lower bounds $lb(q, s_j) = \max_i\{|d_i - \delta(\theta_i, s_j)|\}$. Without loss of generality, let us assume these values are ordered such that $j < \ell \Rightarrow lb(q, s_j) \leq lb(q, s_\ell)$.

During the search procedure, we will have a nearest neighbor candidate s_c (the first one being the closest pivot), and a search radius d such that $\delta(q, s_c) \geq d$ (the first one being $d = 0$).

While $\delta(q, s_c) > d$ we increase d, each time by ϵ (ϵ of the machine) until:

1. either $lb(q, s_j) = d$ for some element s_j, in which case evaluate $\delta(q, s_j)$ and we update our candidate $s_c := s_j$ if $\delta(q, s_j) < \delta(q, s_c)$, and continue to iterate,
2. either $\delta(q, s_c) = d$, in which case we can stop.

The only distances evaluated by this algorithm are those of elements such that $lb(q, s_j) < \delta^*$, therefore it is range-optimal. Thus, if one only considers distance computations for analyzing this algorithm, it is equivalent in complexity to Algorithm 1. Indeed, we observe that the algorithm described above strongly relies on the fact that the only cost involved in search algorithms is the one of evaluating the distance function while any other calculations are negligible. This is not very realistic: even if the distance computation is expensive, the proposed schema performs a huge amount of calculations (not involving distances), making the asymptotic analysis ineffective.

We now consider the extra CPU time for analyzing the described range-optimal algorithm. Then, we propose two variants of this algorithm, the last one being at least as efficient as Algorithm 1.

6.2 Analyzing the range-optimal algorithm

Considering the extra CPU time, the cost of Algorithm 1 is the cost of computing the lower bounds distances (p distances plus $O(n)$ extra CPU time), the cost of sorting the lower bounds ($O(n \log n)$), and the distance computations for the non-discarded objects. Its expected cost is

$$\mathbb{E}(T) = \left(p + n\mathbb{E}\left(\nu(\delta^*)^p\right)\right) \cdot cost(\delta) + O(n \log n) + n\mathbb{E}\left(\nu(\delta^*)^p\right). \quad (5)$$

This means that if $cost(\delta) = o(\log n)$ (reasonable for many practical applications), the most expensive part of the algorithm is the sorting instruction. Now, the question is how to reduce the extra CPU time.

6.3 CPU-efficient range-optimal algorithms

6.3.1 Using linear selection

One alternative to reduce the CPU time is to replace the sort instruction in Algorithm 1 by a linear selection instruction (using the selection in worst-case linear time algorithm, see Cormen et al. [10], Chapter 9.3) on each iteration of the last for cycle. The pseudocode is presented in Algorithm 2.

Algorithm 2 Range-optimal pivot-based NN search algorithm with linear selection

Require: $q \in \mathbb{U}$
Ensure: NN is the nearest neighbor of q in $\mathbb{S}$
1: $mindist \leftarrow \min_{i=1}^{p}\{\delta(\theta_i, q)\}$
2: $NN \leftarrow \theta_{\arg\min_{i=1}^{p}\{\delta(\theta_i,q)\}}$
{Computing lower bound distances}
3: **for all** $s_i \in \mathbb{S} - \mathbb{P}$ **do**
4: $lb(q, s_i) \leftarrow 0$
5: **for all** $\theta_j \in \mathbb{P}$ **do**
6: **if** $|\delta(s_i, \theta_j) - \delta(\theta_j, q)| > lb(q, s_i)$ **then**
7: $lb(q, s_i) \leftarrow |\delta(s_i, \theta_j) - \delta(\theta_j, q)|$
8: **end if**
9: **end for**
10: **end for**
{Searching for NN}
11: **for** $i = 1$ to $n - p$ **do**
12: $s' \leftarrow Select(i, \mathbb{S} - \mathbb{P})$
13: **if** $lb(q, s') > mindist$ **then**
14: break
15: **end if**
16: **if** $\delta(q, s') < mindist$ **then**
17: $mindist \leftarrow \delta(q, s')$
18: $NN \leftarrow s'$
19: **end if**
20: **end for**
21: **return** NN

The complexity of Algorithm 2 is the cost of computing the lower bounds distances (p distances plus $O(n)$ CPU extra time), the cost of the selection ($O(n)$ on each iteration of the last for cycle), and the distance computations for the non-discarded objects. Its expected cost is

$$\mathbb{E}(T) = \left(p + n\mathbb{E}\left(\nu(\delta^*)^p\right)\right) \cdot cost(\delta) + O(n) + O(n) \cdot n\mathbb{E}\left(\nu(\delta^*)^p\right). \quad (6)$$

Therefore, if $n\mathbb{E}\left(\nu(\delta^*)^p\right) = o(\log n)$, Algorithm 2 is more efficient than Algorithm 1. However, it has a worst case of $O(n^2)$ extra CPU time.

6.3.2 Using a heap

A better alternative than using linear selection is to use a heap to check the objects in order, according to their lower bound distance. Note that the heap does not require extra space, as it can be build over the array that stores the lower bound distances. The pseudocode is showed in Algorithm 3.

Algorithm 3 Range-optimal pivot-based NN search algorithm with a heap

Require: $q \in \mathbb{U}$
Ensure: NN is the nearest neighbor of q in $\mathbb{S}$
1: $mindist \leftarrow \min_{i=1}^{p}\{\delta(\theta_i, q)\}$
2: $NN \leftarrow \theta_{\arg\min_{i=1}^{p}\{\delta(\theta_i,q)\}}$
{Computing lower bound distances}
3: **for all** $s_i \in \mathbb{S} - \mathbb{P}$ **do**
4: $lb(q, s_i) \leftarrow 0$
5: **for all** $\theta_j \in \mathbb{P}$ **do**
6: **if** $|\delta(s_i, \theta_j) - \delta(\theta_j, q)| > lb(q, s_i)$ **then**
7: $lb(q, s_i) \leftarrow |\delta(s_i, \theta_j) - \delta(\theta_j, q)|$
8: **end if**
9: **end for**
10: **end for**
{Converting list of objects in $\mathbb{S} - \mathbb{P}$ in a heap}
11: $\mathbb{S}' \leftarrow MinHeapify(\mathbb{S} - \mathbb{P})$
{Searching for NN}
12: **for** $i = 1$ to $n - p$ **do**
13: $s' \leftarrow HeapExtractMin(S')$
14: **if** $lb(q, s') > mindist$ **then**
15: break
16: **end if**
17: **if** $\delta(q, s') < mindist$ **then**
18: $mindist \leftarrow \delta(q, s')$
19: $NN \leftarrow s'$
20: **end if**
21: **end for**
22: **return** NN

The complexity of Algorithm 3 is the cost of computing the lower bounds distances (p distances plus $O(n)$ CPU extra time), the cost of MinHeapify ($O(n)$), the cost of HeapExtractMin on each iteration of the last for cycle ($O(\log n)$), and the distance computations for the non-discarded objects. Its expected cost is:

$$\mathbb{E}(T) = (p + n\mathbb{E}\big(\nu(\delta^*)^p\big)) \cdot cost(\delta) + O(n) + O(\log n) \cdot n\mathbb{E}\big(\nu(\delta^*)^p\big) \quad (7)$$

If $n\mathbb{E}\big(\nu(\delta^*)^p\big) = o(n)$, Algorithm 3 is always more efficient than Algorithm 1 (and it is clearly superior to Algorithm 2). In the worst case, both Algorithms 1 and 3 have the same time complexity.

7. CONCLUSIONS

This paper presented a formal analysis of nearest neighbor search algorithms for pivot-based indexes. We presented a brief probabilistic analysis, and then showed the expected cost for range-optimal search algorithms. We also showed that algorithms based on a fixed-search radius have a probability to fail returning the nearest neighbor, and that range-optimal algorithms are always more efficient if the probability of not discarding an object is a concave function (e.g., in the case of uniform distance distribution). Additionally, we extended our analysis by considering the extra CPU time, and showed that this is important to be able to distinguish between different range-optimal algorithms.

The analysis that considers the extra CPU time showed that the proposed range-optimal algorithms have a complexity of at least $O(n)$, because all of them must compute the lower bound distances for all objects in the dataset. Thus, if the distance function is not expensive (such as the Minkowski distances, which are widely used in vector spaces and are $O(d)$, with d the dimensionality of the space), the extra CPU time cost cannot be neglected. An interesting question that this analysis raises is if there is a range-optimal algorithm for pivot-based indexes that has $O(n^\alpha)$ extra CPU time, with $0 < \alpha < 1$.

Further work could consider extending the analysis of the nearest neighbor search to other indexing schema, like the ones based on partitioning the spaces into zones, and to include the I/O cost in the cost model (for indexes stored in secondary memory). We also observe that while we applied the probabilistic framework to the analysis of the uniform distribution, it would be interesting to do the same for some other distributions. In fact, we consider that determining theoretical distance distributions that are reasonable for the analysis is an interesting topic itself.

8. REFERENCES

[1] R. Baeza-Yates, W. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proc. 5th Combinatorial Pattern Matching (CPM'94)*, LNCS 807, pages 198–212, 1994.

[2] C. Böhm. A cost model for query processing in high dimensional data spaces. *ACM Transsactions on Database Systems*, 25(2):129–178, 2000.

[3] C. Böhm, S. Berchtold, and D. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[4] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proc. ACM International Conference on Management of Data (SIGMOD'97)*, pages 357–368, 1997. Sigmod Record 26(2).

[5] W. Burkhard and R. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, 1973.

[6] B. Bustos, O. Pedreira, and N. Brisaboa. A dynamic pivot selection technique for similarity search. In *Proc. 1st International Workshop on Similarity Search and Applications (SISAP'08)*, pages 105–112, 2008.

[7] E. Chávez, J. Marroquín, and R. Baeza-Yates. Spaghettis: an array based algorithm for similarity queries in metric spaces. In *Proc. String Processing and Information Retrieval (SPIRE'99)*, pages 38–46. IEEE CS, 1999.

[8] E. Chávez, J. Marroquín, and G. Navarro. Fixed queries array: A fast and economical data structure for proximity searching. *Multimedia Tools and Applications (MTAP)*, 14(2):113–135, 2001.

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

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and

C. Stein. *Introduction to Algorithms, Second Edition.* The MIT Press and McGraw-Hill Book Company, 2001.

[11] G. Hjaltason and H. Samet. Ranking in spatial databases. In *Proc. 4th International Symposium on Advances in Spatial Databases (SSD'95)*, LNCS 951, pages 83–95. Springer, 1995.

[12] L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbor approximating and eliminating search (AESA) with linear preprocessing-time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.

[13] G. Navarro. Analyzing metric space indexes: What for? In *Proc. 2nd International Workshop on Similarity Search and Applications (SISAP'09)*, pages 3–10. IEEE CS Press, 2009. Invited paper.

[14] J. Venkateswaran, T. Kahveci, C. M. Jermaine, and D. Lachwani. Reference-based indexing for metric spaces with costly distance measures. *VLDB Journal*, 17(5):1231–1251, 2008.

[15] E. Vidal. An algorithm for finding nearest neighbors in (approximately) constant average time. *Pattern Recognition Letters*, 4:145–157, 1986.

[16] P. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 311–321, 1993.

[17] P. Yianilos. Excluded middle vantage point forests for nearest neighbor search. In *DIMACS Implementation Challenge, ALENEX'99*, Baltimore, MD, 1999.

[18] P. Zezula, G. Amato, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach (Advances in Database Systems).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.