Abstract

This study is focused on the following similarity problem: Given n data points in a high dimensional feature space. Every data point p_i is the center of a hyper-sphere with radius r_i. A query point q is considered to be similar to a data point p_i, if the query point falls within the sphere of p_i. We call such p_i a cover point. We suppose to build an algorithm that returns a cover point if it exists for any given query.

The solutions that were investigated here are essentially reductions that can rely on both of the classical near or nearest neighbor problems. We try several approaches to map our similarity problem to the near/nearest neighbor problem. The first approach relies on dividing the data points with respect to their radii. The second approach named "Separation method" relies on increasing the distances of the data points. A third approach is based on preprocessing data structures that use the spheres of the points. Then based on the closest point to the given query it can return a cover point. The forth approach tries to generalize the Separation approach using intuitions from the similarity.

In practice, we use the near neighbor algorithm E^2LSH [2] for solving the mapped problem. All of our approaches based on E^2LSH have typically sub-linear running time on the data points and perform significantly better than the naive search and the E^2LSH near neighbor algorithm itself. The additional space requirements of our methods are linear, and thus applicable. We also developed a specific version of the Separation method for the l_2 norm and, the unit sphere. This method is the best among our methods for points distributed on the unit sphere. Finally, We generalize the Separation method for every possible l_p norm, including the "fractional norms".

Acknowledgements

First and foremost I would like to thank my thesis advisor, Prof. Shimon Ullman, for his patience, good advice and support during this research.

I would like to thank Prof. Harry Dym, Prof. Ronen Basri and Prof. Edriss Titi for their help. I am very grateful to Alexandr Andoni from M.I.T. for his generous help in answering my questions about his $E^2LSH - 0.1$ package. I am very grateful to Denis Simakov for his helpful comments. I would like to thank my friend Michael Dinerstein for his help in building the figures and reviewing this report. Also I wish to thank my friend Daniel Reichman for his helpful comments. Jasmine Tal and Felix Polyakov deserve thanks for their help in reviewing this report. I am grateful to Dan-Michael Levi and Boris Epshtein for their help with providing parts of the data. Finally, I wish to warmly thank my family and friends from my small city Tamra for their love and support.

In memory of my father Radwan Hegaze.

Contents

List of Figures

1 Introduction

In this work we develop algorithms for searching neighboring points in high-dimensional spaces. The problem is motivated by feature matching in computer vision applications. We describe below related background to the current problem, the motivation coming from feature matching, and define in more detail the search problem we deal with.

A classical similarity search problem is also known as the nearest neighbor search (NNs). It involves a collection of objects which are characterized by a collection of relevant features, and are represented as points in a high dimensional feature space. Given a query object which is also represented as a point in the feature space, the problem is to find efficiently its nearest neighbor point, which represents the most similar object to the query object. This problem was studied and solved for the low dimensional space case, but it remains difficult and challenging for the high dimensional space case. The problem in high dimensional space is of major importance to variety of applications; including information retrieval, statistics and data analysis, data compression, pattern recognition, machine learning, and image and video databases.

Our work is motivated in part by problems arising in recognition and classification in computer vision. The task of visual classification is the recognition of an object in the image as belonging to a class of similar objects, such as a face or a car. To do this we build a database of images relying on a classification approach using a feature based representation. We will consider in particular methods that use image fragments as classification features, but other classification features can be used in a similar manner. The fragments used for classification are selected from a training set of images based on a criterion of maximizing the mutual information between the fragments and the class they represent. They are then stored in a database and used in the classification of new inputs. Each fragment can be considered as a weak classifier that has its own detection threshold, selected so as to achieve the optimal Separation between class and non-class examples (for more details see Ullman et al [24, 25]). The present work focus on the case of binary class vs. non-class classification; further research can extend the approach to multi class classification problems.

During recognition, the fragments database receives a new image (a typical size can be $\sim 1000 \times 1000$ pixels), and it is required first, to find image patches which are similar to the stored fragments. The process starts by constructing from the new image a large set of image patches ($\sim 20 \times 20$ pixels each). An image of size $\sim 1000 \times 1000$ can produce $\sim 10^6$ patches for matching. The patches are created by running a window of size $\sim 20 \times 20$ on the new image. Each possible window defines an image patch, and each new patch is treated as a query image. We then search the database to find the stored fragment that is most similar to the query.

This similarity problem can be translated to a search for neighboring points in a high dimensional attribute space as follows. Each stored fragment becomes a point in the attribute space together with a sphere with radius which is equal to its detection threshold. A query point (patch) is considered similar to a given stored point if their distance is smaller or equal to the point threshold. We will call our problem the $PLDS$ problem, for Point Location in Different Sphere (see definition 1 in Section 3.3). One can look at such a similarity problem as a generalization of the classical similarity problem. Our problem can also be mapped to the problem called 'near neighbor'. In this version, the goal is to report all the points within a given radius R (also called epsilon range search). The mapping is obtained by searching the near points to the query point using a radius equal to the maximum threshold.

For radii that are significantly smaller than the average inter-point distance we can rely on the near neighbor search for solving the $PLDS$ problem, by replacing all the individual radii with the maximum radius in the database, and then running the near neighbor algorithm on search's radius equal to the maximum threshold.

Our research focuses on the case where the thresholds of the points can be large compared with the mean inter-point distance, as often happens with data points, derived from computer vision recognition problems. In particular, the ratio between the mean inter-point distance and the maximum threshold can be smaller than 2. Mapping the problem to the near neighbor search with a radius equal to the maximum threshold gives in this case poor and unstable improvement (if at all) compared with the naive search. By naive search, we mean that for each query we search for the cover point of the query by computing the distances from the query to all the data points one by one. Since in the worst case there is no cover point, then we will compute all such distances, the worst case running time in this case is $O(dn)$, where n is the number of points and d is the dimension.

In such hard cases we develop methods to solve the problem by using new reduction methods to the near/nearest neighbor problem, which will significantly improve the running time compared with the naive search. It seems natural to rely on the near/nearest neighbor in solving the $PLDS$ problem for two reasons. First, as mentioned before, the two problems have a similar formulation as searching neighboring points in high dimensional spaces. Second, the area of NNs search is an active research area, and therefore if we obtained a successful reduction to NNs, then any future improvement in the algorithms of NNs will also improve with it the solution to the $PLDS$ problem.

Since in our applications we do not distinguish between a threshold of a fragment and a radius of a point, we will alternate freely between the two terms.

Most of the algorithms in high dimensions solve the nearest neighbor problem not in an exact manner, but by some approximate solutions e.g. [17, 16, 13, 3, 8]. Most of the cases we will analyze below, the solutions will assume that we use exact near/nearest neighbor algorithm and not approximated near/nearest neighbor algorithm(see definition 4 in subsec. 3.3). There are two reasons for this use of exact NN algorithms. First, the analysis using approximated algorithm is fairly straightforward (for example see appendix B.4), and therefore for simplicity and intuition we prefer the exact algorithm. Second, algorithms have been recently developed which solve the exact near neighbor problem with high probability[11]. Thus, the exact analysis by itself can be used in practical applications.

2 Structure of the Thesis

The rest of this thesis is organized as follows: in the next section we present the background for our problem such as the 'curse' of dimensionality, the fractional norms, and definition for our $PLDS$ problem and other related problems. Then we present a brief overview of the approaches for solving the near/nearest neighbor problem and the $PLDS$ problem. We also present the near neighbor algorithm E^2LSH that we will use in practice. In sections {4,5,6,7} we will present the methods that we used to map the $PLDS$ problem to the NNs problem. The methods are the Direct Multi-level method, the Separation method, the Intersection method and the Similarity with Virtual Levels method, respectively. In the reminder of this report we present experimental results and conclusions.

3 Background

3.1 The Curse of Dimensionality

The curse of dimensionality first defined by Bellman [4], refers to the exponential growth of hypervolume as a function of the dimensionality. In the field of NNs, the term describes the phenomena that the algorithms of NNs and related problems become less and less efficient as the dimension grows. More specifically, their space or time requirements grow exponentially with the dimension.

The failure to remove the exponential dependence on the dimension, led many researches to conjecture that no efficient solutions exist for the exact version of the NNs problem and its related problems, when the dimension is sufficiently large (see [22]). At the same time, it raised the following question: Is it possible to remove the exponential dependence on the dimension, if we allow the answers to be *approximate* (see definition 4 in 3.3). During recent years, some algorithms have been developed that indeed show that in many cases approximation enables a reduction of the dependence on the dimension, from exponential to polynomial, both in their space and time requirements.

3.2 The Minkowski Norms and the Fractional Norms

All the algorithms we consider use a measure of distance between points in metric spaces. A common way to measure such distances is based on l_p norms.

The l_p norm is used to define distance by:

$$dist_p^d(x, y) = \left[\sum_{i=1}^{d} \parallel x^{(i)} - y^{(i)} \parallel^p \right]^{1/p}, \tag{1}$$

where d is the dimension of the space and p is the parameter of the norm. For $p \geq 1$, they are also called the Minkowski norms (during this work we assume that $p < \infty$). These norms were extended by Aggarwal et al [1] for p such that $p \in (0, 1)$, we will call such distance measures the fractional norms. Note that the fractional norms are not even metric distances in the mathematical sense as the triangle inequality does not hold. The reason for this is that the unit sphere under fractional norms is no longer convex (see fig. 1). This extension was motivated by the need to find suitable distance measures for high dimensions, since, Beyer et al [5] show, the nearest neighbor search in high dimensions may be no longer meaningful for most

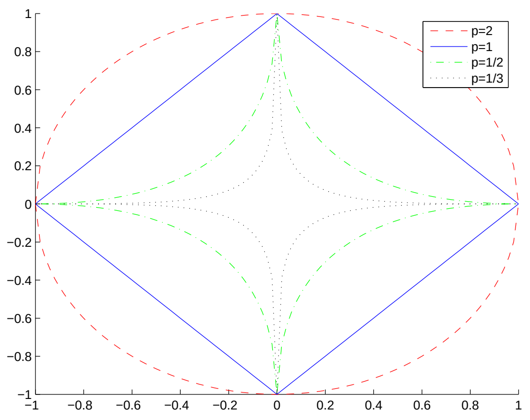

Figure 1: Unit spheres for l_2, l_1, $l_{1/2}$ and $l_{1/3}$ in two dimensions

of the data distributions and distance measures. The reason is, that as the dimension increases, the distance to the nearest and farthest neighbor tend to converge to the same value. Aggarwal et al, showed that using the fractional norms as the distance measure in high dimensionality keeps a satisfactory contrast between the data points. Thus, solving the nearest neighbor problem using a fractional norm gives more meaningful results, also solving clustering problems give more robust clusters. Therefore, the fractional norms started to attract interest see e.g. [15],[10]. We will refer to the fractional norms also as the l_p norms with $0 < p < 1$.

3.3 Definition of the $PLDS$ and Related Problems

In this section we define formally the $PLDS$ problem, and some related problems.

The problem that we are dealing with is the following. A set of n points is given in a space, with a threshold associated with each point. These n points are also called the data set. We need to find an algorithm that efficiently finds for a given query point q in the space a data set point that covers it, or to report that there is no data set point that covers the query point.

The data point is said to cover the query if the distance between the query and the data set point is at most equal to the given threshold of the point. We define our problem formally as follows

Problem definition 1 *Point Location in Different Spheres(PLDS)* [1]
Given n spheres in a d-dimensional space $\mathbf{R}_{\mathbf{p}}^{\mathbf{d}}$, where the distance function in R_p^d is some l_p norm, centered at $P = \{p_1, ..., p_n\}$ with radii $\{r_1, ..., r_n\}$ respectively, preprocess the points in P to efficiently answer the following query: for every query point in $\mathbf{R}_{\mathbf{p}}^{\mathbf{d}}$, if there exist a point $p_i \in P$ such that $q \in S(p_i, r_i)$ (sphere centered on p_i with radius r_i), call p_i a cover point and return it, otherwise return NO.

We will concentrate on the case of l_2 norm since most of the applications use the Euclidian metric as a measurement of the distance.

We next define the Approximate $PLDS$ problem; we will call it the $\gamma - PLDS$ problem.

Problem definition 2 *Approximate Point Location in Different Spheres($\gamma - PLDS$)*
Given $\gamma > 0$ and n spheres in R_p^d centered at $P = \{p_1, ..., p_n\}$ with radii $\{r_1, ..., r_n\}$ respectively, preprocess the points in P to efficiently answer the following query: for every query point in $\mathbf{R}_{\mathbf{p}}^{\mathbf{d}}$, if there exists a point $p_i \in P$ such that $q \in S(p_i, r_i(1+\gamma))$ (sphere centered on p_i with radius $r_i(1+\gamma)$), then return a single point p' such that $q \in S(p', r'(1+\gamma))$, otherwise return NO.

Also we need to define formally the problem of finding nearest neighbor and near neighbor. The nearest neighbor search is defined formally as follows

[1]The definition of this problem is a generalization of the definition of Point Location in Equal Spheres(PLES) problem defined in [16].

Definition 3 *(Nearest Neighbor Search (NNs))*[2]
Given a set P of n points in a space R_p^d, preprocess P so as to efficiently find the point in P closest to a query point q.

The definition generalizes naturally to the case where we want to return $k > 1$ points. The approximate version of the NNs problem is defined as follows:

Definition 4 *(Approximate Nearest Neighbor Search ($\epsilon - NNs$))*
Given $\epsilon > 0$, we say that a point $p \in R_p^d$ is a $(1+\epsilon)$-approximate nearest neighbor of q if $dist(p,q) \leq (1+\epsilon) dist(p^*,q)$, where p^* is the true nearest neighbor to q.

Almost all the algorithms for proximity in high-dimensional spaces proceed by reducing the problem to the problem of finding an *approximate near neighbor*, which is the decision version of the approximate nearest neighbor problem. As an example we mention the algorithms in [13] and [16], that use this reduction to solve nearest neighbor problem. The definition of the near neighbor problem is given by:

Definition 5 *(R-Near Neighbor Problem ($R - NN$)):*
Given a set of points $P \subset R_p^d$ and a radius $R > 0$, construct a data structure to efficiently answer the following: for a query point q, find all points $p \in P$ such that $dist(p,q) \leq R$.

We next define an approximate version of the near neighbor, in this version it is sufficient to return a single approximate near neighbor point.

Definition 6 *(The (R,c) Near Neighbor, or $(R,c) - NN$)*
Given a set P of n points in space R_p^d and two positive constants R and c , design a data structure that supports the following operation: For any query

[2]Although the term NNs is formally for representing the Nearest Neighbor Search problem, during this work we will sometimes mean by NNs both the near and the nearest neighbor search problem.

$q \in R_p^d$, if there exists $p \in P$ such that $dist(p,q) \leq R$, find a point $p' \in P$ such that $dist(q,p') \leq cR$.

3.4 Algorithms for The Near/Nearest Neighbor Problem

Here we represent briefly the known approaches for solving the $NNs(\epsilon - NNs)$ problem. As we mentioned before, the nearest neighbor problem is conjectured to suffer from the curse of dimensionality (see subsection 3.1). Assume that the number of dimensions is d and the number of points is n. In particular, the exact NNs problem has a solution with $O(d^{O(1)}logn)$ query time, using roughly $n^{O(d)}$ space [21]. The exponential dependence of space and/or time on the dimension has been observed in applied setting as well. Many popular data structures using linear or near-liner storage and rely on partitioning the data space (such as KD tree algorithm [7]), exhibit query time linear in n when the dimension exceeds certain threshold (usually 10-20 dimensions), for more information see [26]. Therefore, Most of the algorithms in high dimensions are solving the approximate version of the NNs. Finding solution to $\epsilon - NNs$ for an arbitrary small $\epsilon > 0$, has been studied extensively. Arya et al. [3] obtained an algorithm with query time $O(exp(d) \cdot \epsilon^{-d}logn)$ and pre-processing time $O(nlogn)$. Clarkson [8] obtained a different algorithm which improved the dependence on ϵ in the query time to $O(exp(d) \cdot \epsilon^{-(d-1)/2})$. Kleinberg [17] gave an algorithm with $O(nlogd)^{2d}$ pre-processing and query time polynomial in d, ϵ and $logn$, and another algorithm with pre-processing polynomial in d, ϵ and n but with query time $O(n + dlog^3n)$.

But again the dependence in d for the query time or the space requirements in these approaches is still exponential. The algorithms that have an exponential dependence on d for their query time or space requirements suffer from the curse of dimensionality.

Newer Algorithms were developed using a relatively new approach that based on what called *locally sensitive hashing functions*, these algorithms solve the $\epsilon - NNs$ problem by reducing it to the problem of finding an approximate near neighbors (see definition 6 of $(R, c) - NN$), which is the decision version of the approximate nearest neighbor problem. The first algorithms for $(R, c) - NN$ in high dimensions were obtained by using the technique of random projections. This technique is applicable for the l_p norms, such that $p \in [1, 2]$. The basic locally sensitive hashing functions work for the hamming space $\{0, 1\}^d$, and then embedded into the suitable l_p for $p \in [1, 2]$. We mention here the algorithm of Indyk and Motwani [16], this algorithm uses $O(n^{1+1/\epsilon} + dn)$ pre-processing time and required $O(dn^{1/\epsilon})$ query time for $\epsilon > 1$. An improvement to the query time for the algorithm to $O(dn^{1/(1+\epsilon)})$ were done by Gionis et al [13] for any $\epsilon > 0$, thus its running time was sub-linear. Recently was developed algorithm based on work of Datar et al [11], this algorithm solves the exact version of the Near Neighbor Search. It is based on a new family of hash functions that works directly on l_p^d norm for $p \in (0, 2]$. This algorithm were also implemented to the l_2 norm and the algorithm package was called the $E^2LSH - 0.1$ [2]. We rely on this algorithm for our practical implementation, thus, it represented next subsection. On the other hand, there are only a few studies related to the $PLDS$ problem. One recent example is a study of multimedia identification, which was applied for audio fingerprinting by Goldstein et al [14], for more details and comparing this method with our methods see appendix F.

3.5 Generic locally-sensitive hashing scheme

The description of this subsection and the next one follows [2]. Consider the space R_p^d with the norm l_p, we can use the technique of locally-sensitive hashing [11] to solve the $R - NN$ problem (definition 5). We define the LSH for a domain S of points as:

Definition 7 *A family $\mathcal{H} = \{h : S \rightarrow U\}$ is called locally sensitive, if for* *any query point q and data point v in R_p^d, the function $p(t) = pr_{\mathcal{H}}[h(q) = h(v) : \|q - v\|_p = t]$ is strictly decreasing in t. In other words, the probability of collision of points q and v is decreasing with the distance between them.*

Thus, if we consider any query point q and any data points v, w such that $v \in B(q, R)$ and $w \notin B(q, R)$, then $p(\|q - v\|_p) > p(\|q - w\|_p)$. Intuitively, in the pre-processing we could hash the data points into some domain U, and then processing the query will be by computing the hash of q and consider only the points with which q collides. This way we can find the $R - NN$ points, but it is not necessary that the running time is efficient.

To achieve the desired running time, we need to amplify the "gap" between the collision probabilities for the range $[0, R]$, i.e. where the near neighbor points lie and the range (R, ∞). For this purpose we concatenate several functions $h \in \mathcal{H}$. In particular, define a function family $\mathcal{G} = \{g : S \to U^k\}$ such that $g(v) = (h_1(v), \cdots, h_k(v))$ for some suitable integer k, where $h_i \in \mathcal{H}$. For an integer L, the algorithm chooses L functions $g_1, \cdots, g_L$ from $\mathcal{G}$, independently and uniformly at random. during preprocessing, the algorithm stores each data point v in buckets $g_j(v)$, for all $j = 1, \cdots, L$. To process a query q, the algorithm searches all buckets $g_1(q), \cdots, g_L(q)$. For each data point v found in a bucket, the algorithm computes the distance from q to v, and reports the data point v if and only if, its distance is at most R (i.e. v is a near neighbor).

3.6 Solution Using Exact Near Algorithm E^2LSH

We will represent the algorithm E^2LSH (for Exact Euclidean LSH) , we will use this algorithm as a 'black box' component to solve the NNs problem. This algorithm receives three parameters as input. First, R the radius of searching, such that just points within distances less or equal to R will be returned. Second, Two point files, one includes the query points and the other include the data points. The final input is p which is the success

probability required.
The algorithm E^2LSH available at [2] is based on recent work of Datar et al [11]. The algorithm solves the exact version of the near neighbor search using the technique of Locally Sensitive Hashing (LSH). It is based on a new family of hash functions that works directly on l_p^d norm for $p \in (0, 2]$, without the need to use any embedding, as in the earlier LSH algorithm (see [13]).

E^2LSH solves a probabilistic version of the R-near neighbor problem (see definition 5), which we call a $(R, 1-\delta)-$ *near neighbor* problem. In this case, each point p satisfying $\| q - p \|_2 \leq R$ has to be reported with a probability at least $(1 - \delta)$.

The new locality-sensitive hashing scheme [11] solves the *approximate* version of the R-near neighbor problem, called the (R,c)-near neighbor problem (see definition 6) where $c = 1 + \epsilon$, $\epsilon > 0$. In that formulation, it is sufficient to report with a constant probability *any* point within the distance of at most cR from the query q if there is a point in the data set points within distance at most R from q. For the approximate formulation, the LSH query time is $O(n^\rho)$, where $\rho < 1/c = \frac{1}{1+\epsilon}$. Note that for $\epsilon > 0$ the running time is sub-linear in n.

To solve the $(R, 1-\delta)$ formulation, E^2LSH uses the basic LSH scheme to get all the near neighbors, including the approximate ones, and then drops the approximate near neighbors by a post-processing step (as was mentioned in 3.5). As a result, the running time of E^2LSH depends on the data set points. more specifically, E^2LSH running time depends on the distances of the points from the query point. It is slower for "bad" data sets, e.g., when for query q, there are many points from the data set clustered right outside the ball of radius R centered at q.

In contrast to the original LSH scheme, E^2LSH empirically estimates the optimal parameters k and L (see 3.5) for the data structure, instead of using theoretical formulas. E^2LSH computes the parameters as a function of the data set and optimizes them to minimize the actual running time of query on the host system.

3.6.1 The Estimated Running Time of E^2LSH

Assume R is the search radius of the algorithm, and the data points are at least within distance cR from the query. The estimated running time for the E^2LSH in the worst case is $O(log(1/\delta) \cdot O(n^{\rho(c)}logn + n^{\rho(c)})) \approx O(log1/\delta) \cdot O(n^{\rho(c)}logn)$ where $\rho(c) < 1/c$. This estimation is a coarse estimation but it is sufficient for our purposes, for more details and more accurate estimation for the running time see appendix A.

3.6.2 The Near Method

In this section we describe a "naive" way to use E^2LSH to solve the $PLDS$ problem, which we will call the Near method.

As we mentioned in the introduction above, there is a "naive" way to use any near neighbor algorithm which reports all the points within the search radius, to solve the $PLDS$ problem. This is obtained simply by running the algorithm on radius $R = r_{max}$, where r_{max} is the maximum radius in the database, and then performing a post-processing step, which checks for each point whether it covers the query, and finally returns those points that cover the query.

The running time for the $PLDS$ using the Near method and the E^2LSH is Time(Near Method)=Time(E^2LSH)+Time(post-processing).

The E^2LSH will be used below as a 'building block' within a number of schemes for solving our original $PLDS$ problem. It is therefore useful to note, in summary, the E^2LSH can be applied efficiently with parameters (c, R) provided that only a small fraction of the points lie inside the sphere with radius cR around the query point.

The Near method is not always practical, since the maximum radius could be relatively large. Assume that for the E^2LSH algorithm $R = r_{max}$, and that most of the database points are at distance $(1 + \epsilon)r_{max}$ from the query.

In order to have good performances using the algorithm, we have to insure that ϵ is sufficiently large. Thus, if the maximum radius is relatively large compared with the distances of most of the points in the database we cannot obtain good performances. Therefore, the E^2LSH algorithm for large radii is inefficient.

In the next four sections, we present alternative methods we used for mapping the $PLDS$ problem to the near/nearest neighbor problem which have advantages over the naive Near method. The methods are called the Direct Multi-level method, the Separation method, the Intersection method, and the Similarity with Virtual Levels method.

4 Direct Multi-level Method

The motivation of this method comes from the idea of dividing our original $PLDS$ problem into several subproblems, which we will call 'levels'. Each one of these subproblems has data points which have almost the same radius, i.e. they differ by not more than a γ fraction, where γ is the discrimination parameter which specifies the fraction of mistakes that we allow. We can consider every subproblem as a nearest neighbor problem and solve it independently. In this case we solve the approximate problem, or as we call it the $\gamma - PLDS$ problem (see definition 2).

As we show above (in 3.6.2), we can solve the exact version of the $PLDS$ using a near neighbor algorithm, but the solution may be inefficient in its running time. As we will see the division of the problem to levels can make the running time more efficient, but we will need to estimate the number of levels and the optimal way for division.

The solution obtained by dividing the problem into several subproblems can be considered as a generalization of the near/nearest algorithm itself, since after dividing the original problem into different subproblems, each subproblem will be a new independent near/nearest problem. We can look at the regular near/nearest algorithm as a Multi-level method with the number of levels equals one. Thus, from our point of view, the near/nearest algorithm is just a special case of the Multi-level method. In the case of using a near algorithm, we call this special case the Near method (see 3.6.2).

4.1 Multi-level Using a Nearest Neighbor Algorithm

Using a nearest neighbor algorithm under the Multi-level Method requires a relaxation on the original problem, i.e. it solves the approximate $PLDS$ problem ($\gamma - PLDS$), instead of solving the original $PLDS$ problem.

The basic idea behind this method is to convert the $PLDS$ problem to a Multi-level NNs problem. This is done by discretizing the interval of the thresholds $I = [r_{min}, r_{max}]$, into several intervals $(I_1, I_2, ..., I_k)$. Let us call R_i the maximum threshold in any interval I_i, the value R_i presents the threshold of level i, where $k = \lceil log_{(1+\gamma)}(r_{max} - r_{min}) \rceil$ and γ is the allowed fraction of the solution error.

The threshold in each level is computed as follows: for level i, $R_i = (1+\gamma)^i \cdot r_{min}$. Therefore, the levels' thresholds for all the k levels are $(r_{min}(1+\gamma), ..., r_{min}(1+\gamma)^k)$, where $r_{min}(1+\gamma)^k \geq r_{max}$. We chose to define the levels by a multiplicative factor, rather than using intervals of constant size, since similar for the $\epsilon - NNs$ we are interest in solutions that for any cover point p_i with threshold r_i they provide an approximated cover point p' that its distance is at most $(1+\gamma)r'$ from the query. To achieve the same accuracy using a constant level sizes we will need more levels, since we need to take the size of the smallest level in this scheme as the size for all the levels in the constant size scheme.

Now we divide the data points of the original problem among the different levels such that, any point p_j belongs to level i, if and only if its threshold satisfies the restriction $R_{i-1} < r_j \leq R_i$.

In this way each level out of the k different levels has points with thresholds that have similar values, in particular, their thresholds differ by at most

γ fraction.

Thus, if our applications are not sensitive to γ, then we can consider each level as a separate NNs problem. We can run now a nearest neighbor algorithm on every level, but we should use the following strategy for solving our original problem: on level i, the algorithm will return the closest point if the distance from the query to this point is at most equal to R_i, and nothing otherwise.

The algorithm can stop whenever it finds answer on any level, or may run over all the levels. Thus, the worst case for the running time is when the algorithm tries all the levels. If the algorithm returns point p' that has threshold r', then the distance of the point p' from the query point will be at most $r'(1+\gamma)$. Therefore, this method actually solves the $\gamma - PLDS$ problem.

Note: If we use an approximate nearest neighbor algorithm instead of the exact one, i.e. the one that solves the $\epsilon - NNs$ problem (definition 4), then we have to take the approximation factor ϵ into account too. The reason is that the algorithm in this case can return point p' that is $r'(1+\gamma)(1+\epsilon)$ far from the query. Assume that Γ is the allowed error fraction that we need for our application. In other words, assuming that there exists a cover point for the query point, we need the returned point p' from the Multi-level method to be within distance at most $(1+\Gamma)r'$ (comparing to its threshold r'). To satisfy this distance condition, we need to determine the number of levels by determining γ that gives the required Γ. Easy computations show that $\gamma = \frac{\Gamma-\epsilon}{1+\epsilon}$ guarantees that the final error fraction is at most Γ.

4.2 Multi-level Using a Near Neighbor Algorithm

We recall that the $R-NN$ algorithm should report all the points within any given search radius R. In this case we can look on the Multi-level method as a generalization of the Near method (see 3.6.2), since we will divide the problem into subproblems and we will run the Near method with a suitable search radius on each subproblem. In this way we will solve the exact version of the $PLDS$ problem.

4.2.1 How to Divide

The division of the points into levels here is more flexible than in the case of the nearest neighbor algorithm. At each level we find all of the near points inside the search radius and from them by post-processing step we can find those that cover the query point. As we show in the description of the Near method, this solves the exact $PLDS$ problem. In reality, each level is considered as a subproblem of a $PLDS$ problem.

The running time of the Multi-level method using $R-NN$ algorithm in the worst case is the sum of the running time for all the levels. Assume Tn is the running time of the $R-NN$ algorithm, and Tp is the running time of the post-processing step. Thus the running time for any level i is

$$T(i) = Tn_i + Tp_i. \tag{2}$$

We conclude from above that any possible division of the interval thresholds $[r_{min}, r_{max}]$ for any number of levels can be performed to solve the exact version of the $PLDS$ problem. For most of the distributions of the data points in high dimensionality Tp_i is negligible compared with Tn_i. This is because that the reasonable size of the maximum threshold r_{max} is smaller than the mean distance of the points. Most of the distributions points in high dimensions have equidistant points with small variance [5]. Thus, the number of points that are within distance r_{max} from the query is negligible for most of the distributions of the points. Furthermore, most of the cases the levels that have the largest radii will contain a small number of data points.

We next estimate the best division and the optimal number of levels that will give us the optimal running time for all of the levels of the Direct Multi-level method.

We discuss below the main alternative methods for dividing the points according to the threshold.

4.2.2 Equal Levels Widths

The simplest way for division is to divide the interval equally. We divide the data points to the different levels that have equal widths by their thresholds, i.e. Let us assume that the number of levels is equal to k. First, we need to determine k, then we compute the width of every level call it Δ by $\Delta = \frac{r_{max}-r_{min}}{k}$. The upper bound threshold in each level is $R_1, R_2, ..., R_k$, where $R_i = r_{min} + i \cdot \Delta$, so the lower bound threshold is $R_{i-1} = r_{min} + (i-1) \cdot \Delta$, for any level i. Note that by this definition $R_k = r_{max}$.

For simplicity we assume that the maximum threshold and the minimum threshold are equal to the upper bound and the lower bound thresholds respectively in each level.

Now, for any level i, we can run some $R-NN$ algorithm with search radius R_i. After doing a post-processing step, we can return those points that cover the query point. We can stop the algorithm on any level that returns a positive answer. Therefore, running over all the levels gives the worst case running time.

4.2.3 Power Growth Levels Widths

This division has the property that the widths of the levels are growing as a power of some base b, where $b = (1+\gamma)$ and $(\gamma > 0)$, such that the search radius of level i is $R_i = b^i \cdot r_{min}$, and the width of level number i is $\Delta_i = R_i - R_{i-1} = (1+\gamma)^i \cdot r_{min} - (1+\gamma)^{i-1} \cdot r_{min}$. It is the same technique that we used above, for solving the Direct Multi-level based on a nearest neighbor algorithm (see 4.1).

Unlike the previous case, here we apply an additional heuristic step. Our heuristic is to take a permutation of the original widths, as follows. Assuming that there are k levels and that the original order of the levels widths is $(\Delta_1, \Delta_2, \cdots, \Delta_k)$, we suggest to take the permutation $(\Delta_k, \Delta_k - 1, \cdots, \Delta_1)$ as the new levels widths. The number of levels k will be determined empirically by optimizing the function of the running time of the algorithm for all the different levels. The running time of the $R - NN$ algorithm is usually very dependent on the search radius R and the number of points n. Note that the search radius of the $R - NN$ algorithm equals to R_i the maximum threshold at level i. Using the permutation above means that the levels with the larger radius R will also have a smaller interval, and therefore a smaller number of points in them. Assume that i, and j are any two levels with R_i,R_j being their maximum thresholds and Δ_i,Δ_j being their widths respectively. Assume also that R_i is smaller than R_j. Our heuristic comes from the intuition that taking Δ_i larger than Δ_j in this case improves the running time of the Multi-level method.

4.2.4 Searching for the Best Intervals Division

Here we will use a more general way to find the best division of the radii range to intervals, by searching for it within different possible values for the base interval b and possible values for k that determine the number of levels. We suggest here a way to determine the widths of the levels and the number of the levels independently. by finding both the optimal b and the optimal k will improve the running time of the Direct Multi-level method.

For applying the search we try several relevant values for b (say $b = 0.5$ to $b = 2$, with the step 0.1). We similarly try several relevant values for k (say $k = 1$ to $k = 25$) for each such b. We calculate for the value $\Delta = \frac{r_{max} - r_{min}}{\sum_{i=1}^{k} b^i}$ for each b and for each k. Further, we determine the widths of the levels as follows; $\Delta_k = b^1 \cdot \Delta$, $\Delta_{k-1} = b^2 \cdot \Delta$,...,$\Delta_1 = b^k \cdot \Delta$, where $R_i = r_{min} + \sum_{j=1}^{i} \cdot b^{k-j-1} \Delta$.

Note that if the base $b > 1$ then we will have a power growth levels widths such that $\Delta_i > \Delta_j$ for $i > j$, if the base $b = 1$ then we will have equal levels widths, and if $b < 1$ then $\Delta_i < \Delta_j$. Thus, this divisions generalize the idea of both of the divisions that we have suggested earlier.

4.2.5 The Running Time Using E^2LSH

In practice we will use as a ‘black box‘ the E^2LSH near algorithm which reports the near points with high probability. In the Multi-level case we fix the search radius of the algorithm at each level to be equal to the maximum threshold for the level, i.e. to be equal to R_i for each level i.

As we have shown earlier, the running time for each level is the sum of two parts, Tn the near algorithm time, and Tp the time of post-processing. We assume as above that Tp is negligible compared with the Near time Tn, This is for two reasons. First, the high dimensionality of the points causes that most of the points are equidistant, a reasonable size to the maximum threshold r_{max} is significantly smaller than the mean distance of the points. Second, The post-processing time becomes large when the Near computation finds many irrelevant points within the search radius. This happens with higher probability when the radius R increases. However, by our choice of intervals, the number of points in the intervals usually go down with the radius. As a result, we found that the post-processing time Tp is usually negligible compared with the near neighbor time Tn. We therefore, chose to determine the optimal number of levels by experimental optimization of the running time of the near algorithm Tn for all the levels. The experimental optimization does not require the application of the E^2LSH algorithm, but an empirical evaluation of an equation we derive next for the expected running time.

Assume that R is the radius of search and that the distances of most of the points from the query point are at least $(1+\epsilon)R$. Let $c = (1+\epsilon)$, and the maximum radius R_k of level k equals to r_{max}. The E^2LSH algorithm running time can be estimated as (see subsection 3.6.1)

$$O(log1/\delta) \cdot O(dn^{\rho(c)}logn) \approx O(log1/\delta) \cdot O(dn^{\frac{1}{c}}logn).$$

From our experiments the parameter δ is not affect at all the optimal number of levels, we also plan to use a fixed δ for our experiments. Thus, for optimizing the number of levels we can use the following estimation to the running time.

$$O(dn^{\rho(c)}logn) \approx O(dn^{\frac{1}{c}}logn).$$

We took a coarse estimation for c, such that we took as c, the ratio between the mean inter-point distance and the radius search R. The estimation for the Multi-level can be derived from this estimation as follows. Assume that R_i is the search radius at level i and that the distances of most of the points from the query point are at least c_iR_i. We can then estimate c_i as

$$c_i = \frac{cR_k}{R_i}, \tag{3}$$

where R_k is the maximum threshold in the database and R_i is the maximum threshold in level i. For the k levels we can estimate the running time as

$$O(d\sum_{i=1}^{k} n_i^{\rho(c_i)} \cdot logn_i), \tag{4}$$

where n_i is the number of points at level i that can be computed by simply counting all the points at the different levels given their thresholds. Alternatively, we can estimate the number of points at each level if we can estimate the thresholds distribution of the points.

Optimizing k, the number of levels, cannot be done analytically, but it can be solved experimentally by computing a reasonable number of possibilities and then taking k, the number of levels, to be the one with the minimal estimated running time (a reasonable check is from k=1 to 25 levels). This can be done efficiently using equations (3) and (4) above.

The only value that we have to estimate is the value of ϵ. In most cases it is straightforward task to estimate ϵ for two reasons. First, as was mentioned in [9], the distribution of distances between a query point and the data set points in most cases does not depend on the specific query point, but it is an intrinsic properties of the data set. Second, it was shown in [5] that most data set distributions have distances between the data points that tend to be equal in high dimensions. In other words, the variance of the distances decreases as the dimensionality increases. As discussed further below, the value of ϵ can then be estimated from the variance of the inter-point distance of points chosen randomly in a high-dimensional space.

We found in simulations that the number of optimal levels depends on the following three factors: ϵ, the number of points n and the support of the thresholds distribution. Increasing ϵ causes that the number of levels decreases. On the other hand, increasing n causes the optimal number of levels to increase but it increases very slowly. Increasing the support of the thresholds distribution Increases the optimal number of levels. The number of optimal levels is more sensitive to ϵ than to n. Considering our data points that have minimum threshold 0.1 and maximum threshold of 0.9, For large value of ϵ (larger than 1 for number of data points n ranges from 10000 to 100000 points) the optimal number of levels almost does not depend on the number of the points an it is one. For small values of ϵ the optimal number of levels is more depend on the number of data points n, but it is increasing very slowly when n increases (a reasonable number of levels in this case is ~ 5). A coarse estimation of the number of levels is sufficient to our purposes, since we notice that the running time function is not sensitive to the number of levels k. Thus, one can easily find an estimation of the optimal number of levels for a given ϵ value, that can work almost optimally for different number of data points.

4.2.6 Comparing the Different Divisions Using E^2LSH

The Multi-levels scheme with equal intervals (see 4.2.2) improves the running time compared with the single-level Near method considerably in the cases when the maximum threshold is large, i.e. when ϵ is small. We try also the power law for the interval division (see 4.2.3) for our experiments on data points that have a uniform distribution on the unit sphere. The improvement is slightly better for the algorithm running time compared with the equal interval method, if the distribution of the thresholds is normal. We noticed also that if the number of levels is large then the power width levels are more robust and close to the real optimal. We can find almost the best division by searching the possible intervals divisions (see 4.2.4). But the improvement using the division we find by searching is small and sometimes negligible compared with the previous two schemes. For our experiments, we will not use the search scheme since it will take quadratic running time $O(b \cdot k)$ to the computer to find the optimal division, where k is the level numbers and b is the power of the level width. The previous two methods take linear time of $O(k)$ to find their optimal division, thus we prefer to use them.

Next we will present our second method which is based on reduction from the $PLDS$ problem to the near/nearest neighbor problem called the Separation method.

5 The Separation Method

In this section we suggest a reduction from the $PLDS$ to the near/nearest neighbor problem, by using an additional dimension. The reduction works theoretically for every l_p norm such that $0 < p < \infty$.

The method proceeds by adding a single dimension to the original space. The reduction guarantees that points which cover the query in the original space are included in the ball of radius r_{max}(maximum threshold) centered at the query point in the new space, while points that do not cover the query lie outside this ball. We call this reduction the Separation method.

5.1 Solving the Problem in l_p Norm

Our original similarity problem is as follows: we are given a data set of $P = \{(p_1, r_1), \cdots, (p_n, r_n)\}$, where the p_i are d dimensional points and the r_i are their corresponding radii spheres. A query point q is considered to be similar to a data set point p_i if the point q falls within the sphere of point p_i.

We show that this similarity problem becomes a standard nearest neighbor problem, if we map it to a new problem of $(d+1)$ dimensions, by adding a new dimension for every data point. The value of the new dimension for a point p_i is a function of its radius r_i. This applies to cases where the l_p norm is used as a measurement of the distance, for both the Minkowiski norms $(1 \leq p < \infty)$ and the fractional norms $(0 < p < 1)$ (see 3.2).

More precisely, assume r_{max} is the maximum radius in the data set, then every data point p_i with coordinates $\{p_i^{(1)}, \cdots, p_i^{(d)}\}$ in the original space, will have the coordinates $\{p_i^{(1)}, \cdots, p_i^{(d)}, f(r_i)\}$ in the new space. The query point that originally had the coordinates $\{q^{(1)}, \cdots, q^{(d)}\}$ becomes the point with coordinates $\{q^{(1)}, \cdots, q^{(d)}, 0\}$ in the new space. The $f(r_i)$ is given by,

$$f(r_i) = (r_{max}^p - r_i^p)^{1/p}. \tag{5}$$

Following this embedding, our original similarity problem becomes a classical nearest neighbor problem.

Formally, we define $dist_i$, and $dist'_i$ as the distances between p_i and q in the d dimensional space (R_p^d) and in the new $d+1$ dimensional space $(R_p^{(d+1)})$ respectively.

We can compute $dist'_i$ using $dist_i$ by,

$$dist'_i = \left[\sum_{j=1}^{d+1} \mid p_i^{(j)} - q^{(j)} \mid^p\right]^{1/p} = (dist_i^p + (r_{max}^p - r_i^p))^{1/p}. \tag{6}$$

Lemma 8 *For any l_p norm ($\infty > p > 0$), we use for each point i, the function $f(r_i)$ (5) as a value for the new dimension. If we assume that p_j is a cover point and p_k is a non-cover point, then in the new space $R_p^{(d+1)}$, their distances necessarily satisfy that, $dist'(p_j, q) \le r_{max}$ and $dist'(p_k, q) > r_{max}$.*

Proof. In the original space p_j covers the query q, and p_k does not cover the query q. Thus, $dist_j = dist_i(p_j, q) \le r_j$ and $dist_k = dist(p_k, q) > r_k$.

Now from eq.(6) we have in the $(d+1)$ dimensional space,

$$dist'_j = (dist_j^p + (r_{max}^p - r_j^p))^{1/p},$$
$$dist'_k = (dist_k^p + (r_{max}^p - r_k^p))^{1/p}.$$

Since $dist_j$ is at most r_j and $dist_k$ is at least $(r_k + \delta)$ for some $\delta > 0$,

$$dist'_j \le (r_j^p + r_{max}^p - r_j^p)^{1/p} = r_{max},$$
$$dist'_k > (r_k^p + r_{max}^p - r_k^p)^{1/p} = r_{max},$$

so the claim follows. ■

Corollary 9 *For any query, assume p_n is the nearest neighbor point in the new space $R_p^{(d+1)}$. The distance of p_n satisfies $dist'(p_n, q) \le r_{max}$ if and only if there exists a cover point among the original data set points.*

Proof. It follows directly from lemma 8 above. ■

We succeeded theoretically to cluster the points into two groups, the "good" points are inside the ball with radius r_{max} centered at the query point and the "bad" points are outside this ball (see fig. 2). But in practice, for most of the nearest neighbor algorithms we need to ensure a reasonable distance between the nearest point and the approximated nearest points. Thus, if the distances between the points included in the ball and the points clustered right outside of it are very small, it will be difficult to apply the NNs algorithm and its performances will decrease.

From our analysis above, we conclude that points in the original space R_p^d that have distances from the query point larger than r_{max} will have larger distances in the new space $R_p^{(d+1)}$, hence those points are not problematic. Furthermore, for most of the near/nearest algorithms this property improves the performance of the algorithm. In the same way the points that have originally distances less or equal to r_{max} from the query point, will also have larger distances in $R_p^{(d+1)}$, but their distances are affected by two factors. The first factor is the ratio between r_{max} and $\widetilde{r}$ (where $\widetilde{r}$ is the smallest radius among the points that are originally inside the ball $B(q, r_{max})$). The second factor is the parameter p of the l_p norm. If one of these factors is large, then those points will be close to the boundary of the ball $B(q, r_{max})$ in the new space.

In the following lemma we show the relation between the parameter p of the l_p norm and the distances of the points. More specifically, we will show that the separation between the points included inside the ball $B(q, r_{max})$ and those outside it improves as we decrease the parameter p of the norm l_p. In other words, in general "bad" points get further distance from the query point, and "good" points get closer to the query point as p decreases.

We also show that the critical points for nearest neighbor algorithms are those that are originally included in the ball $B(q, r_{max})$. Such points have large affect on our separation.

Lemma 10 *For any data point p_i (such that $r_i < r_{max}$) in the space R_p^d, if the parameter p of the norm converges to infinity, then the distance of p_i in the new space $R_p^{(d+1)}$, $dist'_i$ (eq. 6) converges to the maximum of $\{dist_i, r_{max}\}$.*

And if p converges to zero ,then $dist'_i$ converges to the value $\frac{dist_i}{r_i} \cdot r_{max}$, where $dist_i$ is the distance of p_i in the original space R_p^d.

Proof. Let us assume that $p \to \infty$,and that $r_i < r_{max}$ (for $r_i = r_{max}$ it is immediate, and it always converges to $dist_i$), assume without loss of generality that $r_{max} \geq dist_i$ (i.e. the points are originally included in the ball

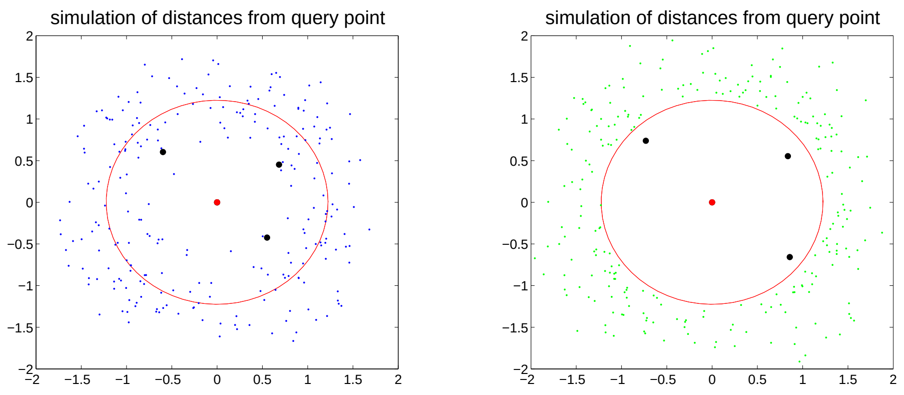

Figure 2: Simulation of distances for 250 points around the query point in l_2 norm. The left plot displayes distances of points in 1024 dimensions of image fragments of size 32×32. The right plot displays the distances of the points after running the Separation method. The circle simulates the ball with the maximum radius, the red point represents the query, the black points represent the cover points and the others are non-cover points.

$B(q, r_{max})$), then

$lim_{p\to\infty} dist'_i = lim_{p\to\infty}(dist_i^p + r_{max}^p - r_i^p)^{1/p} = r_{max} lim_{p\to\infty}(1 + x^p - y^p)^{1/p}$,
where $x = \dfrac{dist_i}{r_{max}} \leq 1$,$y = \dfrac{r_i}{r_{max}} < 1$, we can write it as
$r_{max} \cdot e^{\frac{1}{p} lim_{p\to\infty} ln(1+x^p-y^p)} = r_{max} \cdot e^0 = r_{max}$.
since $lim_{p\to\infty} ln(1 + x^p - y^p) < ln(2)$.

For the second part $(p \to 0)$

$$lim_{p\to 0} dist'_i = lim_{p\to 0}(dist_i^p + r_{max}^p - r_i^p)^{1/p} = e^{lim_{p\to 0}\frac{1}{p}\cdot ln(dist_i^p + r_{max}^p - r_i^p)}.$$

Now by using Lhopital's rule on the power of e we can continue the analysis...

We used Matlab to calculate the limit above, without any conditions on $r_{max}, dist_i$ and r_i and got that the limit is $\frac{dist_i}{r_i} \cdot r_{max}$. ■

The lemma above shows that in spite of the success to Separation implied by the clustering, this clustering is getting worse as the parameter of the norm p

increases. We conclude from the lemma and from our empirical simulations, that points that are originally outside the ball (q, r_{max}) will increase their distances if we decrease the parameter p of the norm l_p. Furthermore, for the points that originally have distance less or equal to r_{max}, we conclude that if the parameter p of the l_p norm satisfies that $0 < p < 1$ i.e. if we use the fractional norms, then the separation between points inside the ball $B(q, r_{max})$ and those that are outside it, is satisfactory also for large ratios between r_{max} and $\widetilde{r}$.

For p increasing beyond 1, we start to notice that the separation is getting worse. From our observation, we notice that when p increases the points are getting closer and closer to the boundary of the ball $B(q, r_{max})$ from both sides. If p becomes very large then this reduction is not practical any more for nearest neighbor algorithms (see fig. 15). But for near neighbor algorithms it is still practical, especially since it is known that for most of the data points distributions there is a small fraction of points that have distances less or equal to r_{max} in high dimensional spaces, for a reasonable size of r_{max} (see [5]).

We conclude that using the fractional norms give us better clustering between the "good" points and the "bad" points, for solving the $PLDS$ problem. This conclusion gives another evidence to Aggarwal et al's work [1] showing that using fractional norms as distance measures in high dimensions gives more meaningful nearest neighbors search. Here we showed that using fractional norms is also more robust for solving our problem the $PLDS$ problem as it make more robust clustering. We hope also that the large distances between the points inside the ball $B(q, r_{max})$ and those outside it using the fractional norms, will improve sufficiently also the running time of our methods compared with using the Euclidean norm. In this work we will not try in practice our methods under the fractional norms since to our knowledge there is no algorithm for the NNs problem that works for the fractional norms in practice. But theoretically, the E^2LSH algorithm can be implemented for such norms. Therefore, we expect that in the future we will have algorithms for fractional norms that works in practice.

For our present applications, we are interested in particular in the l_1 norm ($p = 1$) where the performance of the reduction is satisfactory, and the l_2 norm ($p = 2$), where the performance is worse than in l_1 norm, but still good enough for the cases where the ratio between r_{max} and $\widetilde{r}$ is not very large.

We conclude that in the cases where the ratio between r_{max} and $\widetilde{r}$ is very large, and a nearest neighbor algorithm is to be used, it will be helpful to divide the problem into several levels, such that at every level we will have suitable ratio between the radii (see 5.3).

when using the near neighbor algorithm, the situation will be more robust and it will not face such problems, in general.

Next we will describe the Separation algorithm that rely on the Separation method.

5.1.1 The Separation Algorithm

- Pre-processing:

1- Run the Separation method above and reduce the problem from d dimensional $PLDS$ problem to $(d+1)$ dimensional NNs problem. By substituting $(r^p_{max} - r^p_i)^{(1/p)}$, in the new dimension for every data point p_i.

- Query processing using a nearest neighbor algorithm
  Extend the query to $(d+1)$ dimensions by substituting zero in the new dimension. Run the NNs algorithm on the data points on the new space. If the returned point have distance less or equal to r_{max} then return it as answer (based on cor. 9), otherwise return '*No* cover points in the data set'.

- Query processing using a near neighbor algorithm
  Extend the query to $(d + 1)$ dimensions by substituting zero in the

new dimension. Run the $R - NN$ algorithm on the new points on radius r_{max}, return the points that returned by the near algorithm as an answer (based on lemma 8). If no points returned, return '*No* cover points in the data set'.

We described above how the Separation method works, and discussed its properties. Next we will analyze the running time of E^2LSH algorithm using the Separation method.

5.1.2 Time Analysis of the Separation Method using E^2LSH

If we assume that R is the search radius of the algorithm, then the running time of E^2LSH is estimated as $O(1/\delta)O(dn^{\rho(c)}logn)$, where we assume that most of the points are at distance not less than cR (see 3.6.1).

- The running time of the Near method is
  Time(Near method)=Time(E^2LSH)+Time(post-processing)
  $=O(1/\delta)O(dn^{\rho(c)}logn) + Time(post - processing)$, where $\rho(c) < \frac{1}{c}$.

- The running time of the Separation method is
  Time(Separation method)=$O(1/\delta)O((d+1)n^{\rho(c_2)}logn)$, where $\rho(c_2) < \frac{1}{c_2}$ and $c_2 > c$.

- In general c_p for the l_p norm is increasing as p decreases.

Note that the fact that $c_2 > c$ improves the running time of the Separation method comparing to the Near method. For more details and to estimate c_2 (or c_p for any l_p norm) see appendix B.1.

We analyzed above the running time of the algorithm using the Separation method. We also compared the running time to the "naive" Near method running time. Next we will address a special application of the Separation method for data points that are distributed on the unit sphere.

5.2 The Separation method for the Unit Sphere

Here we address a special case Separation method applied specifically to data points that are distributed on a sphere and using the l_2 norm. We will restrict the discussion to the unit sphere.
In our similarity applications coming from computer vision, we are in fact interested in points distributed on the unit sphere. The similarity measurements for our data in computer vision problems is generally the Normalized Cross Correlation (NCC). Translating this to the classical l_2 norm causes the data points to be mapped to the unit sphere. We need such a translation since most of the algorithms that deal with nearest neighbor search or similarity assume the use of the Minkowski norms as a distance measurement. In particular, most of the algorithms use the l_2 norm.

Thus, we will analyze here a special Separation method for the case of a data set distributed on the unit sphere, and the l_2 norm as our distance measurement.

Assume p is some data point with threshold r, and q is a query point in the original space R_p^d. Consider the Separation method above (see section 5) then we can look on the points $p' = \{p, f(r)\}$, $p'' = \{p, 0\}$, and $q'' = \{q, 0\}$ as the vertices of a triangle in the new space $R_p^{(d+1)}$ (see figure 3). Note that this triangle does not necessarily have Euclidian geometric properties under any l_p norm, we use it just to make our motivation to the Separation on the unit sphere method clearer.

Assume that the data set points are in a d dimensional l_2 normed space, then the unit sphere centered at the origin lies on $(d-1)$ dimensional space. Our data points that are distributed on the unit sphere lie on a $(d-1)$ dimensional space.

We can take advantage of this fact and instead of adding a new dimension as we do in the general case (section 5), we can use the space R_2^d itself for re-mapping the data points and achieve separation. We will also use the fact that R_2^d is a Hilbert space. First we define a re-mapping of a boundary point

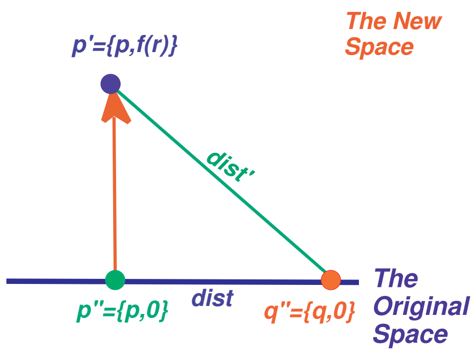

Figure 3: The general Separation method. The red point is the query point q represented in the new space as q'', the green point is a data point p represented in the new space as p'', and the blue point p' represents the new position of the data point p after applying the Separation method.

p_i. Consider the two-dimensional plane π_{p_i} that contain the following three points: the query q, the boundary point p_i itself, and the origin $\overrightarrow{0}$ (see figure 4). Note that the geometric properties (the edges lengths, and the values of the angels) are invariant under rotation, i.e. are invariant for any query point on the unit sphere that are within a distance r_i from the data point p_i. We assume that $(0 < r_i < 2)$, thus γ shown in figure(4) is in the interval $(\frac{\pi}{2}, \pi)$, $\gamma < \pi$ since $r_i < 2$, and $\gamma > \frac{\pi}{2}$ since if $\gamma = \frac{\pi}{2}$ then α should be zero.

The Separation method in this case can be applied by multiplying every point p_i that lie on the unit sphere by a suitable value that we call $ratio_i$. $Ratio_i$ is a function of the radius of p_i and the maximum radius.

This Separation method has the same properties of the general Separation method, in the sense that points that cover the query will lie inside of the ball $B(q, r_{max})$, and points that do not cover the query will lie outside this ball. In addition, *any* boundary point p_i that lies originally on the boundary of the ball $B(q, r_i)$, i.e. its distance from the query equals its radius, will lie on the boundary of the ball $B(q, r_{max})$, after applying the method.

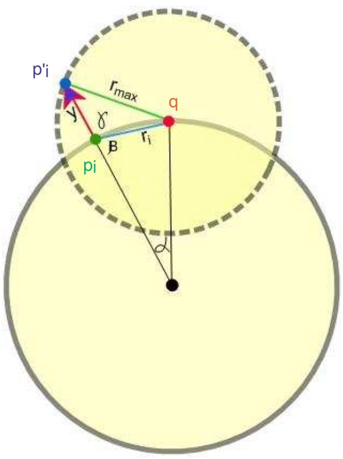

Figure 4: The Separation method for the unit sphere applied for a boundary point. This figure shows the two-dimensional plane π_{p_i}. The black point is the origin, the red point is the query point q, the green point represents a boundary point p_i, and the blue point represents the new position of the boundary point p'_i after applying the Separation method.

To achieve the properties above, $ratio_i$ should satisfy the following: assume that $dist_{io}$ is the distance from the origin to the point p_i. Since the R_2^d space for the l_2 norm is a Hilbert space, we can compute the angle α from figure (4), using the following cosine rule on the triangles

$$r_i^2 = 1 + 1 - 2 \cdot cos(\alpha)$$

The new distance from the origin $dist'_{io}$ given by

$$dist'_{io} = dist_{io} + y = 1 + y. \tag{7}$$

Note that,

$$r_{max}^2 = 1 + dist'^2_{io} - 2dist'_{io} \cdot cos(\alpha). \tag{8}$$

Thus,

$$ratio_i = \frac{dist'_{io}}{dist_{io}} = 1 + y = \frac{2 \cdot cos(\alpha) + \sqrt{(-2 \cdot cos(\alpha))^2 - 4(1 - r_{max}^2)}}{2}.$$

The multiplication of any boundary point p_i by its corresponding ratio $ratio_i$ increases its distance from the origin by $ratio_i$. This causes the boundary point to lie on the boundary of the ball $B(q, r_{max})$, i.e. the distance of the point from the query point will be r_{max}.

The same computation is suitable for non-boundary points i.e. for points that their distance from the query is not equal their thresholds. For any point p_k we first assume that p_k is a boundary point and then compute its ratio. Relying on this assumption, the use of the ratio computed this way satisfies the request that the points that cover the query will lie inside the ball $B(q, r_{max})$, and points that do not cover the query lie outside the ball $B(q, r_{max})$.

We recall that a data point p_k is considered to be a 'cover' point to the query q if q is inside the ball $B(p, r_k)$, and 'non-cover' point if the query is outside that ball.

Lemma 11 *For a query q, applying the Separation method causes that any point p_k on the unit sphere which is originally cover the query lies inside the ball $B(q, r_{max})$, while if p_k is originally a non-cover point for q then it will lie outside $B(q, r_{max})$.*

Proof. See appendix B.2. ■

The above lemma implies that if $dist_k = r_k$, then $dist'_k = r_{max}$ (boundary point case), and if $dist_k$ is smaller or larger than r_k, then $dist'_k$ is smaller or larger than r_{max}, respectively.

This method performs better than the general Separation method using the l_2 norm, for points that are distributed on the unit sphere (see experiments in section 8). This is because, we do not need to add a new dimension to the original points. Furthermore, the new distances from the query for both the cover points and the non-cover points are somewhat larger than the distances in the case of the general Separation method. Increasing the distances improves the running time of the E^2LSH algorithm. Another advantage of using this method compared with the general case Separation method is that it is intuitively simpler and easier to observe.

5.3 Combining Multi-level and the Separation Method

We can combine both the Multi-level approach and the Separation method together for the original data points. Such combining improve the running time of the Direct Multi-level method, for more details see appendix B.3.

We suggested above a reduction form the $PLDS$ problem to the near/nearest neighbor problem, which works for every l_p norm. The method accomplishes that all the points that cover the query point are within distance r_{max} from it, and all the points that do not cover the query have distances greater than r_{max} from it. We also showed how we can use the near neighbor E^2LSH algorithm to solve the $PLDS$ problem, and how this reduction can improve the running time of E^2LSH. Next, we introduce another way that uses a reduction from the $PLDS$ problem to the NNs problem called the Intersection method.

6 The Intersection Method

The intersection method is a reduction of the $PLDS$ problem to the nearest neighbor problem, which also uses information from the balls of the data points. Suppose that q is a query point, and p_i is a data point that covers q within its radius r_i, p_i is the point we want to find. Assume that the nearest neighbor to q is a different data point, denoted by N. If we search the nearest neighbor of q, we will retrieve N rather than p_i. The essence of the intersection method is to create a data structure, using a pre-processing stage, by which the point N will 'point' to the correct point p_i. The method uses two basic facts:

1- If the nearest point has a distance greater than r_{max} then no point covers the query.

2- If there is a point p_i that covers the query and has radius r_i, then the point p_i with radius $2 \cdot r_i$ must also cover N, the nearest point to the query.

This is true because $dist(p_i, N) \leq dist(p_i, q) + dist(q, N) \leq 2 \cdot r_i$. Therefore, fact (2) above assumes that the distance's measurement satisfies the triangle inequality.

6.0.1 Description of the Algorithm

In the pre-processing step we save for each data point p_k a list of data points that are candidates for being cover points for any query that return p_k as its nearest neighbor, call it $list_k$. The candidates associated with a point p_k are all the data points that cover the point p_k if we double their thresholds (see figure 5 in this figure $p_k = N$). In the query step, we first use a nearest neighbor algorithm to find the closest point N to the query q, then we can find a cover point if it exists by scanning the candidates list of N and computing their distances from the query. If we did not find a cover point among the candidates list of N, we conclude that there is no cover point in the database for the given query.

For the pseudo-code of the algorithm see appendix C.1.

6.0.2 Time Analysis

We next examine the running time of the intersection method when we use a nearest neighbor algorithm.

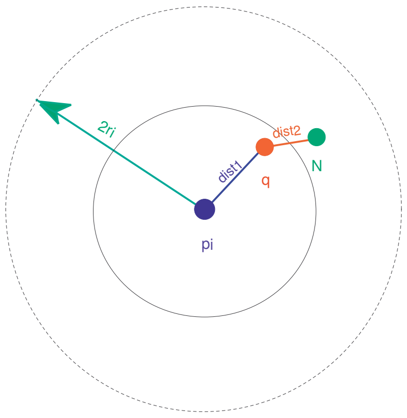

Figure 5: The point q is the query, N is its nearest neighbor, and p_i is a cover point to q with radius r_i.

The pre-processing step running time is $O(dn^2)$ time, since in order to build the list of any data point p_i in the database we need to run over all the other data points in the database and compute their distances from p_i.

Call the NNs algorithm that we use A, the query processing time is equal to the time that the algorithm A takes to find the nearest neighbor point. In addition, we need to take into account the time to go through the data structure of the nearest neighbor point. Assume that the size of $list_i$ that corresponds to the point p_i is S_i. If $S_{max} = \{maxS_i : 1 \leq i \leq n\}$, then in the worst case the time of the query is $T = Time(A) + d \cdot S_{max}$.

Note that by using this method we can retrieve all of the points that cover the query point, since all of them must be included in the list of the nearest neighbor point.

The algorithm is efficient if S_i is sufficiently small compared with n (i.e. $S_i << n$) for most of the points in the data set, namely most of the data points have thresholds that are at least twice smaller than the mean inter-point distance.

6.0.3 Space Analysis

We will discuss the space requirement for the Intersection method in addition to the space used by the nearest neighbor algorithm.

Our pre-processing stage requires in the worst case $O(n^2)$ space, in addition to the space used by the nearest neighbor algorithm. This space is used for building the data structure *list* for all the data set points. This is the worst case requirement, when each data point is included in all of the other points extended spheres. Therefore, it saves all of the data points in each list of the data points. In such a case, the running time algorithm is worse than the naive search and it is better to use another approach. The query processing step in this case includes running over all the n points and computing all the distances from the query point.

In the average case, the space requirements by the intersection method is $O(n)$, since the space that the data structure $list_i$ of point p_i uses should be $O(S_i)$, if we assume as above that S_i is the size of $list_i$. Thus, all the data set points need $O(\hat{S} \cdot n)$ space, where $\hat{S}$ is the average list size and $\hat{S}$ is assumed to be small compared with n.

6.0.4 Using the E^2LSH Algorithm

The intersection method above uses a nearest-neighbor computations. For the other methods we have used in practice the E^2LSH, which as a near neighbor algorithm. A question that arises is whether we can use the same E^2LSH also as an efficient nearest neighbor algorithm?

We can find the nearest point by using a binary search on radius of search R of the $R-NN$ algorithm E^2LSH. The Intersection method is efficient using a $R-NN$ algorithm if we run it on data points with a meaningful NNs, namely, there is a sufficient contrast between the distances of the data points from the query point. In particular, the closest point distance from the query is small compared with most of the points and the maximum threshold. Most of the meaningful data sets in high dimensions are clustered data set

points or data points with implicitly low dimensionality[5]. For more details see appendix C.2.
In the next section, we will present our fourth and last method for reducing the $PLDS$ problem to the NNs problem, called the Similarity method.

7 Similarity Method with Virtual Levels

The basic idea of the similarity method is to use an additional dimension to increase the similarity (or reduce the relative distance) between the query and points with large radius. Points with a large radius are more likely to be missed by a search method that looks for near neighbors, compared with points with a small radius. The similarity method tries to offset this 'disadvantage' of points with large radii. This is obtained by a way of normalizing the distances relative to the radii: the distances of the points from a potential query increased for all the points, but the additional distance for each point is relative to its radius, therefore, points with small radii will increase there distances more than points with large radii. This method is based on generalizing the idea of the Separation method, and considering the l_2 norm.

Another way of looking at the approach is as a tradeoff between distance and radius: We search for the data point that is as close as possible to the query point in the original space, and at the same time has the largest possible radius. The tradeoff between the original features and the new radius feature can be changed by a parameter α. The value of the additional dimension $(d+1)$ that we add to the points is $\alpha \cdot r_i$ for each point p_i in the data set, and $\alpha \cdot r_{max}$ for the query point. Determining the value of α is critical for the algorithm, since α is the parameter that controls the tradeoff between the original distance and the effect of the radius. If α is very large, then the property of the radius in the new space $(R_p^{(d+1)})$ becomes important, and the most similar points to the query are those points with the maximum radius, even if the points are not close enough to cover the query point. On the other hand, if a small α is chosen, then the additional dimension has a negligible effect on the similarity. Therefore, the most similar points in this case are those closest to the query in the original space. It is clear that these points do not necessarily cover the query point.

Ideally, we would like the selected α to satisfy one of the following properties:

- The first cover point must lie within the most similar k points in the new space ($R_p^{(d+1)}$), for some constant k with a high probability.
- The cover points must lie inside some ball in the extended space with a radius R (for some R) centered at the query point, and there are no non-cover points inside this ball.

We chose to focus on the second property because this property gives us the opportunity to use a near neighbor algorithm.

Unfortunately, the requirement above cannot be satisfied efficiently by just substituting a suitable parameter α for the additional dimension. The requirement forces α to be a function of the radius r_i for any point p_i. This complicates the problem of similarity for the new space $R_p^{(d+1)}$, since we will have a different parameter α_i for each different data point p_i. Theoretically this implies that the query point should have in its added dimension the parameter $\hat{\alpha}$, which should equal all the α_is at the same time. Obviously, this requirement cannot be satisfied, and therefore, we should build for each different radius in the data set a special query, and then run all of these queries instead of the single original query. However, this will be of course a highly inefficient solution to our problem.

To avoid the problem above we make a discretization of the thresholds interval values $[r_{min}, r_{max}]$ to k intervals with $(R_1, ..., R_k)$, as the maximum thresholds in all the intervals, and then producing for each original query, k new queries. Each new query corresponds to one interval from the set above. The value of the additional dimension of query number i corresponding to interval number i has the value $\alpha(R_i) \cdot R_i$. We will call this extension the *Virtual Levels*.

We presented above the motivation and a basic use of the Similarity method. We will show examples how we can apply the notion above, by producing some queries in the new space $R_p^{(d+1)}$ to each single query in the original space R_p^d. This extension is called the Virtual Levels and is presented in appendix D.1.

8 Experiments

The experiments below were done using the following methods: the Near method, the Direct Multi-level method, the Exponential Direct Multi-level method, the Separation method (both one level and Multi-level), the Separation method for the unit sphere (both for one level and Multi-level) and the Similarity with Virtual Levels method. The difference between the Direct Multi-level method and the Exponential Direct Multi-level method is in the way we divided the levels. The divisions were the equal widths division and the power growth division, respectively (see 4.2).

Our experiments were done on synthetic data points that we produced randomly on the unit sphere. The reason that we did not use a real data set from vision problems is that currently we do not have a sufficient number of fragments for our experiments. For most of the experiments we produced 128 dimensional data points which were uniformly distributed on the unit sphere. Producing d-dimensional points distributed uniformly on the unit sphere can be done efficiently by picking the d different coordinates as i.i.d. random variables distributed according to the standard normal distribution $N(0, 1)$, and then normalizing the vector to have a unit length [23]. For the Intersection method we used clustered data points on the unit sphere which we also produced synthetically. We chose points with 128 dimensions compatible with the number of dimensions commonly used by the $SIFT$ descriptor [20] which is a useful representation for matching image fragments.

In all of our experiments we used the E^2LSH algorithm as our $R-NN$ algorithm. As we mentioned before this algorithm find the near points with probability $(1-\delta)$. We set the percentage of false negatives that we can tolerate up to 10% (i.e. $\delta = 0.1$). Consequently, at least 90% of the points that cover the queries will be detected by the E^2LSH. In practice, for the "hardest" data points of the Separation method on the unit sphere for example, the E^2LSH algorithm miss 81 cover points from 1000 cover points. Therefore, 91.9% of the cover data points were detected in the worst case. In addition, we present below, the worst case running time since we assumed that we always run over all the levels. Running all over the levels will find all the cover points for any given query. However, If it is sufficient to return a single cover point for each query, then the average running time is sufficiently larger using the Multi-levels methods, since then we can stop the search in any level we find a cover point. We also did not take into account the post-processing time for the methods that need a post-processing step, since in the case of uniformly distributed data, the post-processing time is negligible compared with the time of E^2LSH. If we run our methods on a clustered data, then the post-processing time may be larger. in this case, the Separation method offers an advantage, since it does not need a post-processing step (see sec. 5).

The first experiment is for points uniformly distributed on the unit sphere with 128 dimensions (figure 7). The points are almost equidistant. To estimate this distance, assume in general points distributed on a sphere with radius a in d dimensions, and μ, σ_d^2 are the mean distance and the variance, respectively. We can estimate the mean distance μ and the variance σ_d^2 for points uniformly distributed on sphere as the following $\mu = \sqrt{2}a$ and $\sigma_d = \frac{a}{\sqrt{2d}}$ where d is the dimensionality of the data points [18]. Thus, we estimate the mean distance of points distributed on the unit sphere as $\sqrt{2}$, and σ_d for our data is $\sigma_d \approx 0.0625$. The points thresholds were taken to be

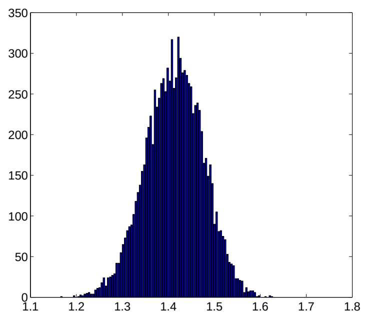

Figure 6: Distances between a query point and 10000 data points with 128 dimensions. The points are uniformly distributed on the unit sphere.

normally distributed with a mean value that equals 0.5 and a variance that equals $\sigma_{th}^2 = 0.01$ ($\sigma_{th} = 0.1$). Thus, the maximum threshold r_{max} is some value that generally belongs to the interval $[0.9, 1]$; the minimum threshold r_{min} usually belongs to the interval $[0.1, 0.2]$. These are somewhat challenging assumption, since the thresholds can be relatively large compared with the inter-point distances.

If most of the points are at distance of no less than $cr_{max} = (1+\epsilon)r_{max}$ from the query, then we can estimate the running time of the E^2LSH as $O(dn^{\frac{1}{c}})$(see 3.6.1). We recall here that the E^2LSH does not need c as an input, but we will estimate it here to make our discussion clearer. We need also to estimate coarsely it c for optimizing the number of levels for the Multi-level methods. But in reality, we used the ratio between the mean distance and the maximum threshold as an estimation for c for optimizing the number of levels. From our observation, we can estimate the distribution of distances of the data points from a typical query, as a normal distribution (figure 6). Therefore, We compute cr_{max} as follows: $cr_{max} \approx \mu - 3\sigma_d = 1.226$. The number of data points varied in this experiment from 10000 to 100000 with steps of 10000. For this experiment we tried 100 different queries. The running time is the average running time of those 100 different queries.

For the methods that use a single level, we use the maximum threshold

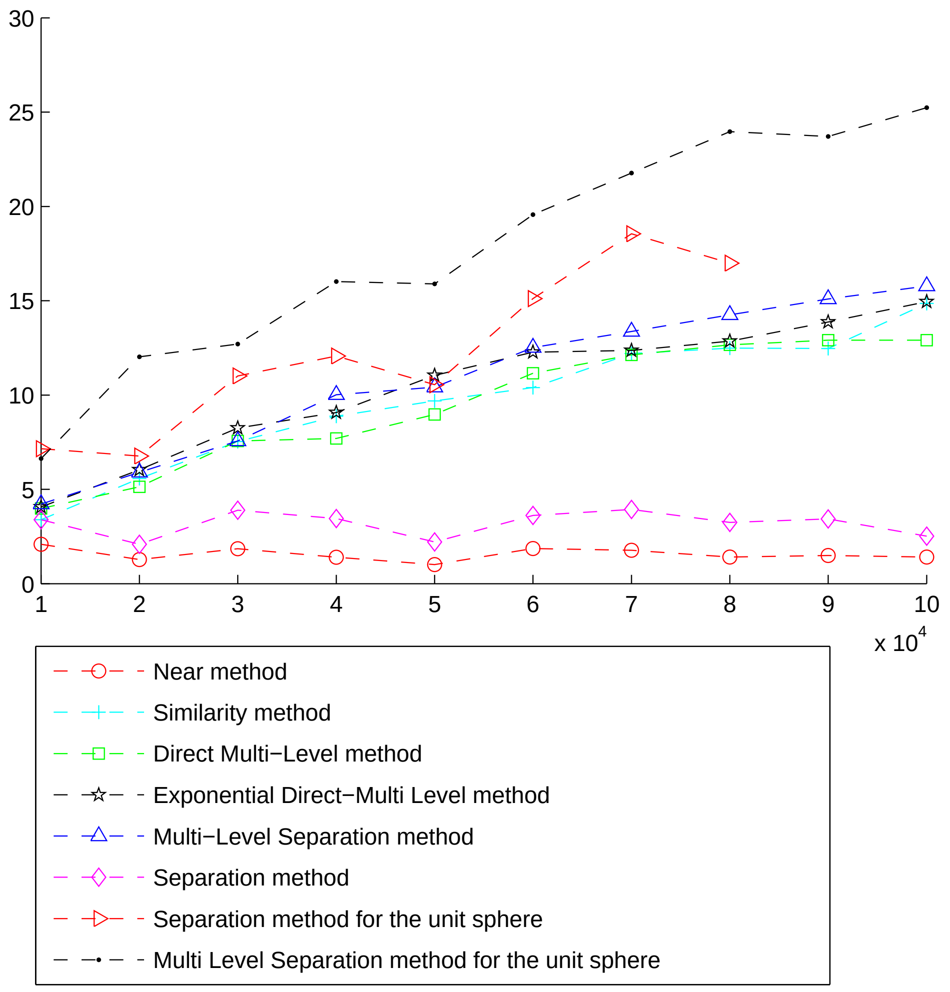

Figure 7: The running time for several methods compared with the naive search. The x axis represents the number of points, and the y axis represents the improvement compared with the naive search. The points have 128 dimensions, uniformly distributed on the unit sphere and have thresholds that are normally distributed according to N(0.5,0.1).

as the search radius for the E^2LSH algorithm. These methods are consequently more affected by the value of the maximum threshold. Most of the non-monotonic 'zigzag' changes in the result graph are because of the changing value of the maximum threshold; these changes are more drastic and noticeable for the methods using a single level. The methods that use Multi-level succeed in making this zigzag changes less noticeable. thus, multi levels are more robust when we use the E^2LSH algorithm.

We see that the direct use of the standard Near search is only slightly better than a naive search that simply test all the points in the data set. The simplest form of the separation method performs only twice better than the Near method. The Multi-level version of the same method performs significantly better than the single level version. The other methods also provide a significant advantage, reducing the running time by a factor of to $15 - 25$, for 100000 points with 128 dimensions. We can also see that the relative advantage increases systematically with the number of points, this is due to the fact that the E^2LSH has a typically sub-linear running time.

There are other factors that can amplify the non-monotonic zigzag changing phenomena. One is the fact that E^2LSH is a randomized algorithm which chooses its hash functions randomly causing some choices to be more (or less) successful for our specific data set points and queries. The most important factor is that the E^2LSH algorithm attempts to approximate the optimal number and the optimal widths of its hash functions in its pre-processing step, depending on both the data set points and the queries set (see [2]). This optimization step is done within a memory bound, and this bound sometimes causes inaccurate optimizations. An example of unsuccessful optimization of E^2LSH because of the memory bound was noticed using the Separation method for the unit sphere for data sets with $90,000$ and $100,000$ points, we therefore removed these two points from the figures.

The second experiment was similar to the first one; the only difference was that we fixed an upper bound to the maximum threshold, which we took to be 0.9. Since $\sigma_d = 0.0625$; thus the ratio $cr_{max} = (1+\epsilon)r_{max} \approx \mu - 3\sigma_d = 1.226$ as before, and therefore $c = 1.363$. The running time was computed by averaging the running time of 100 different queries. The main observed difference in the result was that the changes for the methods that uses a single level were more smooth and robust, since the maximum threshold was bounded (figure 8). The non-monotonic change for the Multi-levels Separation method for the unit sphere is since the optimal number of levels for this method is small (two or three levels). If the optimal number of levels is small and the interval of the radii ($[r_{min}, r_{max}]$) is changing with the number of points then this cause non-monotonic changes for the running time. This changes are more noticeable since we divided the interval according to the power growth levels widths (see 4.2.3).

The third experiment was as follows. The number of the points was fixed to $50,000$, and they are uniformly distributed on the unit sphere, as in the first experiment. The thresholds were normally distributed, and we varied the average threshold from 0.1 to 0.9 with steps of 0.2 and with a variance of 0.01 ($\sigma_{th} = 0.1$). The running time is the average time for answering 100 different queries.

In this experiment we examined the improvement of our methods compared with the Near method (using the basic E^2LSH) and the naive search, by varying the average threshold of the normally distributed thresholds (see figures 9,10). For the applications coming from computer vision, we are interested in the case where the average threshold equals about 0.5. For this average value we have sufficient improvement compared with both of the Near method and the naive search. The case of the average thresholds being equal to 0.9 is unrealistic and represents a very difficult case, since the maximum threshold in this case is generally more than 1.2 (the mean distance of the points is $\sqrt{2} \approx 1.4$); however most of our methods are slightly better than the naive search even in this case. On the other hand, for the average threshold equals to 0.6, the Near method is already worse than the naive search.

The fourth experiment is the same as the first experiment, but the thresholds are uniformly distributed in the interval $[0.1, 0.9]$ and the points are uniformly distributed on the unit sphere. We noticed that the performance of our methods in this case are twice as bad as the case of the normally distributed thresholds (see figure 11).

As we note in appendix F the improvement of our methods for the running time increases with the dimensionality. We call this phenomena the 'benefit of dimensionality'. The reason for such a phenomenon is that the algorithm of E^2LSH is sensitive to the distances of the data points from the query point. The distances of the points turn to be more and more uniform as function of the dimensionality, in other words the variance of the distances σ_d^2 gets smaller as the dimensionality increases [5]. Consequently, for very high dimensions the improvement of the running time because of this phenomena becomes less noticeable. One can notice the improvement in running time that this phenomena causes for the E^2LSH algorithm. Our methods further amplify this improvement, such that it can become very helpful in real applications. Next, we experimented with data points distributed uniformly on the unit sphere with 500 dimensions. We used in this experiment our Separation method on the unit sphere (see figure 12); the number of points was varied from 10000 to 50000 points, and the running time was computed as the average of 100 different queries. The result show that the improvement in running time was about twice compared with data points with 128 dimensions (figure 8). In this case $r_{max} = 0.9$ and $\sigma_d \approx 0.031$, therefore $cr_{max} \approx \mu - 3\sigma_d \approx 1.32$ thus $c \approx 1.466$, note that this is larger than c in the case of 128 dimensions, where it was 1.363.

The final experiments used the Intersection method. The points in this case were clustered on the unit sphere, such that each cluster contains 100 points. Such clustered data points can arise naturally in practical applications. We varied the number of clusters from 100 to 1000 clusters. The clusters were built by 1000 points uniformly distributed on the unit sphere with thresholds that are normally distributed according to $N(0.4, 0.1)$, an upper bound of 0.8 and a lower bound of 0.2. For each cluster we produced points by the following procedure. We randomly chose a point from the cluster from the points that we already computed. We then computed a random vector with a distance from this point distributed according to $N(0.4, 0.1)$, with an upper bound of 0.5 and a lower bound of 0.2. The new point is the normalized sum of the chosen point and the random vector. The diameter of each cluster is at least one. We examined two distributions of thresholds for the same data points set. In figure (13) the thresholds of the points are found from the distribution $N(0.1, 0.1)$ in the same way (with upper bound 0.8 and a lower bound 0.2, thus most of the thresholds have value in the interval $[0.2, 0.3]$); the maximum threshold in the data set is 0.7. In figure (14) the thresholds where chosen from the distribution $N(0.5, 0.1)$ with upper bound equal to 0.9 and lower bound equal to 0.01. The running time was computed by 100 different queries such that 10% of the queries were chosen randomly on the unit sphere, i.e. generally they are not within any cluster, and the remaining 90% of the queries were chosen to be included within some cluster, by choosing the query in the same way as we choose the data points included in each cluster. We also took to account the post-processing step for the Intersection method. The improvement in the running time is significant compared with the Near method and to the naive search. The post-processing step could be large for clustered data. The Separation method have advantage over the other methods in that it does not need any post-processing step. In the first experiment we chose small radii (the average in the interval $[0.2, 0.3]$) compared with the maximum threshold (0.7). The Separation method benefit from this distribution of thresholds. For example, the Separation method for the unit sphere has improvement of ~ 60 times better than the naive search. This improvement was less dramatic when the average threshold was 0.5. There is an important advantage of the Intersection method over all the other methods, in that its improvement is not depend very much on the thresholds distribution, if we satisfy the request that most of the data points thresholds are at least twice smaller than the mean inter-point distance. consequently, assume that the inter-point distance is at least twice larger than the maximum threshold. If all the thresholds equal the maximum threshold the Intersection method will still have an efficient running time. On the other hand all the other methods running time will not be better than the running time of the Near method.

9 Conclusions

We examined in this work the problem of $PLDS$, which is a variation of the near-neighbor and nearest-neighbor problems studied in the past. For these previous problems, efficient methods have been developed in the past, which improve significantly the search time compared with the naive search approach. In practice, this solutions are for the approximated version of the problems and/or solves the problems with constant probability. The solution for the $PLDS$ problem (definition (1) subsection 3.3), in cases where the spheres are significantly small compared with the average inter-point distance, can be obtained efficiently enough by directly using a near neighbor algorithm, which reports every point in any given radius (we called this the Near method). The search will use a near-neighbor algorithm with search radius equal to r_{max}, the maximal radius of the stored data points. When the spheres are relatively large, then the solution using a near neighbor algorithm is inefficient, sometimes even compared with the naive search.

Our aim in this work was to solve more efficiently the $PLDS$ problem, especially for relatively large spheres. The problem is motivated in part by problems arising in computer vision classification, where different data points may have different radii.

Our solutions are, in general, reductions from the $PLDS$ to the classical near and nearest neighbor problems, which work well in both theory and practice. In the section below we briefly list the solutions we have developed and their main properties.

A simple and natural extension of previous near-neighbor methods is the approach we called the Direct Multi-level method. The idea is to divide the problem into several sub-problems with a suitable accuracy such that we can solve every sub-problem independently. Within each sub-problem the search radius is assumed to be fixed, and therefore we no longer confront the problem of different search radii. As it turns out, a simple application of the Multi-level method with fixed intervals does not produce a significant improvement. We therefore developed a more efficient variation of the method that divides the range of radii in an optimal manner.

The second solution, which works theoretically for every norm l_p, for the Minkowski norms ($p \geq 1$), and for the fractional norms ($0 < p < 1$) (see subsection 3.2), is to separate the data points by their thresholds and their distances from the query into two groups. The "good" points, which cover the query point will become included inside the sphere with a radius equal to the maximum threshold, centered at the query point. The "bad" points, which do not cover the query, will end up outside this sphere. We therefore call this method the Separation method.

The third method is to build a data structure for each data point, where we store during a pre-processing step all the candidates to cover the point in question. Relying on finding the nearest neighbor to the query, we can find all the cover points to the query from the data structure that corresponds to the nearest point. This method is called the Intersection method. The fourth method is based on a similar idea to the Separation method; it was developed to generalize the Separation method and to improve its running time for the l_2 norm. The basic idea underlying this method is to rely on similarity indexing in high-dimension feature space, so that the radius sphere of the data points will be considered as a new feature for comparing points.

This is implemented by discretizing the thresholds interval and Separating the data points to the different intervals by adding distances related to their thresholds in a new dimension. During search, we build from the original query point a new query, one for each discrete interval. This was called the Similarity with Virtual Levels method.

All the methods described above rely on the availability of a near-neighbor or nearest-neighbor computation. We next discuss the relationship between the $PLDS$ methods and specific types of near and nearest neighbor algorithms.

If we use a nearest neighbor algorithm for solving the $PLDS$ problem, then the solution by the Direct Multi-level and the Virtual Levels methods are applicable to the approximate version of the problem ($\gamma - PLDS$ see definition 2). In contrast, the solution using the Separation and the Intersection methods are applicable to the exact version of the $PLDS$ problem. The use of a near neighbor algorithm solves the exact version of the $PLDS$ problem for all the different methods. The solution based on the Intersection method cannot directly use the near-neighbor algorithm, unless the near neighbor algorithm is used to find the nearest neighbor, by a binary search (see table (1) for summary of our conclusions).

The discussion so far assumed the availability of a near-neighbor algorithm, but did not depend on a particular form of the algorithm. In practice, we used in all our simulations the near neighbor algorithm known as E^2LSH for solving the mapped $R - NN$ problem. Next, we will analyze how the different methods improve the running time for solving the $PLDS$ problem.

The E^2LSH running time is not affected by the distribution of the data points. The main factor that determine its running time is the distances of most of the data points from the query, compared with the radius of search (We called this ratio c). The running time of this algorithm can be estimated approximately as $O(dn^{1/c})$, where n is the number of points and d is the dimension. The main advantage of our methods over the Near method

Method	A nearest neighbor algorithm solves	A near neighbor algorithm solves	Special assumptions
Direct Multi-level method	$\gamma - PLDS$	$PLDS$	
Separation method	$PLDS$	$PLDS$	
Separation for the unit sphere	$PLDS$	$PLDS$	Works just for points distributed on the unit sphere and the l_2 norm
Intersection Method	$PLDS$	Does not fit	1-Most of the thresholds of the data points are at least twice smaller than the mean inter-point distance
			2- It improve the running time compared with a $R - NN$ algorithm just for clustered data points or implicitly with low dimensionality.
Similarity method with Virtual Levels	$\gamma - PLDS$	$PLDS$	For a NNs algorithm, the minimum threshold is sufficiently large compared with the width of the levels.

Table 1: Summary of the methods.

(using directly the E^2LSH) is that our methods increases the effective value of c. Increasing the value of c is obtained based on two factors. First, by reducing the size of the search radius for large number of points, as done by all of the Multi-levels methods, and by the Intersection method. Second, by increasing the distances of the points from every potential query, as done by the Separation methods, and the Similarity with Virtual Levels. We can attribute each improvement in the running time to one of the two factors above, or to both, as the Multi-level Separation methods uses both of the above factors.

We next discuss how the different methods use these two factors. It is obvious that the first factor is used by the Multi-levels scheme applied to the Direct Multi-level, the Separation, and the Virtual Levels methods. The Intersection method is also based on the first factor, by assuming that the NN point distance is sufficiently smaller than most of the other points, and consequently choosing a small search radius compared with the maximum radius. The methods that rely on the second factor uses the 'gap' between the maximum threshold and the thresholds of most of the data points to increase the distances of most of the points from any potential query. In reality, both of the factors above depends on the 'gap' between the maximum threshold and the average threshold. In the extreme case, all of the radii will be equal to the maximum threshold. In this case, all of the methods except the Intersection method will not perform better than the Near method. Thus, we conclude that the support of the radii distribution is important to increase the ratio c. From the analysis above and from the experiments, the running time of the three methods mentioned is better for radii distributions that are similar to normal distribution than 'harder' radii distribution such as uniform distribution (see figures 8 and 11).

From the analysis above we conclude that the most important factor for the running time is the value of the ratio c. Thus, it is not important if the data points are clustered or not, or if the data points lie on a unit sphere or within the volume; if the distribution of the points provides a sufficient ratio c, then it will run sufficiently fast compared with the naive search. The clustered data set are important for the Intersection method, since it provide the necessary assumption to improve the intersection running time. This assumption is that the nearest neighbor is usually sufficiently closer to the query than the other data points. The Separation methods shows advantage over other methods on clustered data, since the post-processing step could be large for clustered data, and the Separation methods does not need any post-processing step like the other methods.

In contrast to most of the algorithms that suffer from the 'curse of dimensionality' (see subsection 3.1) in high dimensions, the use of the E^2LSH algorithm under our methods gives us an important benefit in high dimensionality. As was noted in section 8 the improvement of the running time of our methods increases with the dimensionality for the same radii distribution. The dimensionality of the data points also serves to improve the ratio c. It is known that all of the data points distributions tend to have more equidistant distances between the data points as the dimension increases [5], and therefore the distances of the close data points to the query increases. Thus, the ratio between the radius search and the distances of the points c increases. This improve the running time of the E^2LSH algorithm, but the improvement using our methods is more noticeable.

We present next the performances of the different methods using the E^2LSH. Assume that for our solution it is sufficient to find a single cover point to the query. Thus, running on all over the levels for the methods uses the Multi-level scheme is the worst case running time. The improvement in the worst case by using the Direct Multi-level method was sufficient compared with the naive search. We noticed that there is no reasonable way of dividing the data points according to their thresholds that can case a drastic improvement compared with dividing the thresholds interval equally. The improvement using the Separation method for the l_2 norm was almost twice better than the Near method, this improvement is insufficient by itself for our purposes, we therefore combined the Separation method with the idea of dividing the problem into different intervals, resulting in what we called the Multi-level Separation method. Using this combined method we got an additional improvement over the Direct Multi-level method. We also developed a special Separation method for the case where the data points are distributed on the unit sphere (as our vision data points are distributed on the unit sphere). We obtained in this case a significant improvement: even without using division into intervals, the performance was better than the Multi-level Separation method. If we also apply the division strategy to the Separation method on the unit sphere, the method becomes more robust and it gives better performances. We called this combination the Multi-level separation method for the unit sphere. We conclude from our experiments that this combined method was the best method for data points distributed on the unit sphere (normalized data points).

The Intersection method, which uses E^2LSH as an implicitly nearest neighbor algorithm, improved the running time of the problem compared with the Near method (E^2LSH itself), if the data points that were used have a meaningful nearest neighbor to the query. Namely, if the closest point has a significantly smaller distance than the majority of the other data points in the data set. In high dimensionality, data points with a meaningful nearest neighbor naturally occur for clustered data points, or for data points lying on lower dimensional sub-space[5]. For our experiments we used clustered data points, and obtained consistent and significant improvement compared with the basic E^2LSH algorithm (see results in sec. 8).

Our final method, the Similarity with Virtual Levels using E^2LSH, also improved the running time, but in most cases the improvement was not better than the Direct Multi-level method, and sometimes even worse.

In all the cases where we divide the data points into intervals, we take the number of levels that empirically optimizes the running time of the algorithm, based on an estimation of the worst case running time. The query running time relying on E^2LSH for a reasonable thresholds sizes is sub-linearly related to the number of the data points in all of the methods. The space requirements were at most linear in the number of data points n, if we consider the additional space needed by the different methods (without the space needed by E^2LSH). The space requirement for the Intersection method is $O(n^2)$ in the worst case, but when the Intersection method is in fact practical, then its space requirement is only $O(n)$.

Next, we present the benefit of the Separation method using fractional 'norms' and propose future approaches related to the fractional norms. The solution that the Separation method obtained for the fractional norms was better than the solution for the Minkowski norms, since the distances between the good points and the bad points becomes larger in general when the parameter of the norm p gets smaller. As a result, we obtain better Separation between cover and non-cover points for the fractional norms. This solution is important for two reasons. First, it has been shown that the fractional norms serve as better distance measurements for high dimensions compared with the Minkowski norms for the nearest neighbor and the clustering problems [1], and we have shown here that this is the case also for the $PLDS$ problem. Second, the algorithms for the NNs improve in term of running time if we increase the distances of most of the data points from the query point. Consequently, the fact that non-cover points increase their distances from the cover points and the query, can improve the running time of the algorithm. A future research direction of interest would be to check whether the Separation method using the fractional norms, with similarity relying on a suitable NNs algorithm that supports such norms, will have a better running time compared with the Separation method using the Minkowiski norms. Such an algorithm can be from the family of LSH methods, which are known theoretically to support the use of any fractional norm (see [11]).

References

[1] C. Aggarawal, D. Keim , and A. Hinneburg. On the surprising behavior of distance metrics in high dimensional spaces. *Proceedings of the International Confererence on Database Theory*, pp. 420-434, 2001.

[2] A. Andoni and P. Indyk. E^2LSH-0.1 User Manual. *avilable at* http://web.mit.edu/andoni/www/LSH/index.html

[3] S. Arya, D. Mount, N. Netanyahu, R. Silveman, and A.Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimension. *Proc. 5th ACM-SIAM SODA*, 1994. Extended version appears as University of Maryland technical report CS-TR-3568, December 1995.

[4] R. Bellman. Adaptive Control Processes: A Guided Tour. Princeton, NJ, Princeton University Press, 1961.

[5] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft: When is 'nearest neighbor' meaningful?, *Proceedings of the 7th International Conference on database Theory*, pp. 217-235, Jerusalem, Israel, 1999.

[6] C. Burges, J. Platt, and S. Jana. Distortion discriminant analysis for audio fingerprinting,*IEEE Transaactions on Speech and Audio processing*,vol 11, No. 3, pp. 165-174, 2003.

[7] V. Castelli. Multidimensional indexing structures for content-based retrieval. IBM Research Report, IBM Thomas J.Watson Resarch Center, Yorktown Heigts, NY.

[8] K. Clarkson. An algorithm for approximate closest-point queries. *Proc. 10th ACM symp. on Computational Geometry*, 1994.

[9] P. Ciaccia, M. Patella and P. Zezula. A cost model for similarity queries in metric spaces. In *Proceedings of PODS'98*, pp. 59-68.

[10] G. Cormode, P. Indyk, N. Koudas, and S. Muthukrishnan. Fast mining of massive tabular data via approximate distance computations. *Proc. 18th Internatinal Conference on data Engineering(ICDE)*, pp. 605, 2002.

[11] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. DIMACS *Workshop on Streaming Data Analysis and Mining*, 2003.

[12] R. Fergus, P. Perona, A. Zisserman. Object class recognition by unsupervised scale-invariant learning. *Proc. of the IEEE Conf. Comp. Vis. Pattern Recog*, pp. 264-271, 2003.

[13] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Peoceeding of the 25th VLDB Conference*, Edinburgh, scotland, 1999.

[14] J. Goldstein, C. Platt, J. C. Burges. Indexing high dimensional rectangles for fast multimedia Identification. *Technical Report*, MST-TR-2003-38 , Microsoft Research, Microsoft Corporation , 10/28/2003.

[15] P. Howarth and S. Ruger. Fractional distance measures for content-based image retrieval.*27th European Conference on Information Retrieval (ECIR, Santiago de Compostela, Spain, Mar 2005)*, pp. 447-456, 2005.

[16] P. Indyk,and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. *STOC 98*, Dallas, Texas, USA.

[17] J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. *Proc. 29th ACM Symposium on Theory of Computing*,pp. 599-608, EI Paso, Texas, USA, 1997.

[18] Al. Lehnen G.E. Wesenberg. The Sphere Game in n Dimensions. *avilable at* http://matcmadison.edu/alehnen/sphere/hypers.htm

[19] K. Lin, H.V. Jagadish, and C. Faloutsos. The TV-tree: an index structure for high-dimensional data. *VLDB*, 3, pp. 517-542, 1994.

[20] D. Lowe. Distinctive image features from scale-invariant keypoints. Int. J. Comp. Vis. 60(2), 91-100 , 2004.

[21] S. Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106:286-303, 1993

[22] M. Minsky and S. Papert. *Perceptrons.* Cambridge, MIT Press, 1969.

[23] J. Poland. Three Different Algorithms for Generating Uniformly Distributed Random Points on th N-Sphere. *avilable at* http://www-alg.ist.hokudai.ac.jp/ jan/randsphere.pdf

[24] S. Ullman, and E. Sali. Object classification using a fragment-based representation. In: *Seong-Whan Lee, Heinrich H. Bulthoff, Tomaso Poggio (eds.). Biologically Motivated Computer Vision, First IEEE International Workshop, BMVC 2000*, Seoul, Korea, May 15-17, 2000.

[25] S. Ullman, E. Sali, and M. Vidal-Naquet. A fragment-based approach to object representation and classification. In: *A. Arcelli, L.P. Cordella and G. Sanniti di Baja (eds.), International Workshop on Visual Form*, Berlin: Springer, pp. 85-100, 2001.

[26] R. Weber, H.J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. *Proc. 24th Int. Cof. VLDB*, pp. 194-205, 1998.

[27] V.M. Zolotarev. One-dimensional stable distributions. vol.65 of Translations of Mathematical Monographs, Americam Mathematical society, 1986.

A The Running Time of E^2LSH

We estimate below the running time of E^2LSH algorithm. The technique used in the algorithm is suitable to any norm l_p in the interval $p \in (0,2]$, however, current implementations of the algorithm apply the technique only using the l_2 norm (for more details and analysis see [11]). First we present the running time for the algorithm using the basic LSH scheme for solving the $(R,c)-NN$ problem (definition 6 in subsection 3.3). For a domain S of a set of points with distance measure D, an LSH family is defined as follows:

Definition 12 *A family $\mathcal{H} = \{h : S \rightarrow U\}$ is called (r_1, r_2, p_1, p_2) sensitive for D if for any $v, q \in S$*

- *if $v \in B(q, r_1)$ then $Pr_{\mathcal{H}}[h(q) = h(v)] \geq p_1$,*
- *if $v \notin B(q, r_2)$ then $Pr_{\mathcal{H}}[h(q) = h(v)] \leq p_2$*

In order to have a description for how a LSH family can be used to solve the $(R,c)-NN$ problem with constant probability see [11]. The running time for the basic scheme for solving the $(R,c)-NN$ problem in constant probability is:

- Consider L is the number of hash functions used and k is the width of each hash function (see L, k in subsection 3.5). For any l_p such that $p \in (0,2]$ there exists an algorithm for $(R,c)-NN$ under R_p^d which uses $O(dn+nL)$ space, with query time $O(dL)$ for the distance computations and $O(dkL)$ for the evaluations of the hash functions, where $L = n^\rho$, $\rho = \frac{ln1/p1}{ln1/p2}$ and $k = log_{1/p_2}n$.
- For the l_2 norm assume that R is the search radius for the algorithm, and most of the points are within distance at least cR from the query point. The query time is $O(dn^{\rho(c)}logn)$, where $\rho(c) < 1/c$ (for $c \in (1,10]$ the inequality is strict).

We can use the basic scheme of LSH to solve the $R-NN$ problem (definition 5, subsection 3.3), for more details see subsection 3.6. The solution for the $R-NN$ problem and the l_2 norm is called E^2LSH. Next, we present an estimation to the running time for the E^2LSH that is sufficient for our purposes.

- As we mention before the E^2LSH can fail to find any point included in his radius of search R with probability δ. Consider L is the number of hash functions used and k is the width of each hash function, the running time of the E^2LSH algorithm is $O(dkL) + O(dL)$ where $L = \lceil \frac{log\delta}{log(1-p_1^k)} \rceil$. The k value is experimentally optimized by the E^2LSH algorithm to minimize the running time (for more details see [2]). For our analysis we will take the value of $k = log_{1/p_2}n$ above from the LSH basic scheme for solving the $(R,c)-NN$ problem. For this value of k we estimate L as $(\frac{log(1/\delta)}{p_1^k})$ (since $log(1+x) \approx x$ for small xs), thus the estimated running time is at most

$$O(dkL) + O(dL) \approx O(dk \cdot \frac{log(1/\delta)}{p_1^k}) + O(d\frac{log(1/\delta)}{p_1^k})$$
$$= O(log(1/\delta) \cdot log_{1/p_2}n \cdot d(\frac{1}{p_1})^{log_{1/p_2}n}) + O(log(1/\delta) \cdot d(\frac{1}{p_1})^{log_{1/p_2}n})$$

  Note that $(\frac{1}{p_1})^{log_{1/p_2}n} = e^{\frac{ln1/p_1 \cdot lnn}{ln1/p_2}} = n^{\frac{ln1/p_1}{ln1/p_2}}$, thus will have the estimation for the running time,

$$O(log(1/\delta) \cdot dn^{\rho}log_{1/p_2}n) + O(log(1/\delta) \cdot dn^{\rho})$$
$$= O(log1/\delta) \cdot O(dn^{\rho}log_{1/p_2}n + dn^{\rho})$$

  Note that in the worst case the running time of the E^2LSH is very similar to the running time of the basic LSH for solving the $(R,c)-NN$ problem.

In practice for the E^2LSH package and by special assumptions on the distribution of picking the hash functions the running time can be estimated in the worst case more accurately by

$$\begin{aligned} &O(log(1/\delta) \cdot d\sqrt{n^\rho} log_{1/p_2} n) + O(log(1/\delta) \cdot dn^\rho) \\ &= O(log 1/\delta) \cdot O(d\sqrt{n^\rho} log_{1/p_2} n + dn^\rho) \end{aligned} \tag{9}$$

Such that computing p_2 in the l_1 norm is as follows: $p_2 = 2\frac{tan^-1(r/c)}{\pi} - \frac{1}{\pi(r/c)} ln(1 + (r/c)^2)$. Computing p_2 for the l_2 norm is as follows: $p_2 = 1 - 2norm(-r/c) - \frac{2}{\sqrt{2\pi}r/c}(1 - e^{-(r^2/2c^2)})$, where $norm(.)$ is the cumulative distribution function (cdf) for a random variable that is distributed as $N(0, 1)$. the value of p_1 can be obtained by substituting $c = 1$ in the formulas above (for more details see [11] and [2]). We use eq.(9) to make more accurate optimizations for the number of levels for the Multi-level methods.

B More Analysis for the Separation Method

B.1 Time analysis restricted to E^2LSH

We will analyze the running time of the algorithm E^2LSH when we run it on our reduced data using the Separation method. We will compare the running time using the Separation method against the Near method (3.6.2).

Note that the E^2LSH algorithm running time is sensitive to the distances of the data set points from the query point(see 3.6.1). If we guarantee that most of the points are within distance at least cR from the query point (assuming $c = (1 + \epsilon)$ for $\epsilon > 0$), then the algorithm will run in time at most $O(log 1/\delta)O(dn^{\rho(c)} logn)$, on radius of search equal to R, where $\rho(c) < \frac{1}{c}$ (see appendix A).

The Separation method improves the E^2LSH algorithm running time, since it makes the distances from the query in the $(d + 1)$ dimensional space larger than the original distances in the d-dimensional space.

Assume that we can guarantee that most of the points in the d-dimensional space are within distance of at least $(1+\epsilon)r_{max}$. For *any* point p_i within distance $dist_i$ from the query point in the original space, we will have in l_2 norm the distance $dist'_i = \sqrt{dist_i^2 + (r_{max}^2 - r_i^2)}$ in the new space (see eq.(6)).

Thus most of the distances of the points in the new space will guarantee that, $dist'_i = (1+\epsilon+\xi_i)r_{max}$, where $1+\epsilon+\xi_i = \sqrt{(1+\epsilon)^2 + 1 - \left(\frac{r_i}{r_{max}}\right)^2}$ (note that for the l_1 norm $\xi_i = 1 - \frac{r_i}{r_{max}}$).

After running the Separation method reduction, we will run the E^2LSH algorithm on radius $R = r_{max}$ on the data points in the new space (R_2^{d+1}), as we mentioned in the algorithm above (see 5.1.1). This way we find all the points that covers the query in the original space (R_2^d), since they must lie inside the ball of radius $R = r_{max}$ in the new space (R_2^{d+1}).

We can assume that the majority of the points have radii around some average, call it the average radius $\hat{r}$. This is very natural assumption since in most real problems the radii are normally distributed or similar to normally distributed. Our data from real computer vision problems satisfies this assumption. Relying on this assumption most of the data set points will have distances that can be estimated as $dist' = c_2 r_{max} = (1+\epsilon+\hat{\xi})r_{max}$, in the new space ,where

$$c_2 = 1+\epsilon+\hat{\xi} \approx \frac{dist'}{r_{max}} = \sqrt{(1+\epsilon)^2 + 1 - \left(\frac{\hat{r}}{r_{max}}\right)^2} \qquad (10)$$

Hence, the query running time of the Separation algorithm will be $O(1/\delta)O((d+1)n^{\rho(c_2)}logn) \leq O(1/\delta)O((d+1)n^{\frac{1}{c_2}})$.

From eq.(6) and the above analysis

$$c_p = 1+\epsilon+\hat{\xi} \approx \frac{dist'}{r_{max}} = \left((1+\epsilon)^p + 1 - \left(\frac{\hat{r}}{r_{max}}\right)^p\right)^{1/p} \qquad (11)$$

for any norm l_p ($0 < p < \infty$). Note that similar to the E^2LSH algorithm there theoretically exists an LSH algorithm based on the same hashing scheme [11] for every norm l_p such that $p \in (0, 2)$. From Lemma(10) and our analysis of the Separation method the distances from the query of the non-cover points are getting larger as the norm's parameter p decreases. Thus, c_p from eq.(11) is getting larger as p decreases, Thus, the ratio of the running time improvement is getting larger as the norm's parameter decreases. Intuitively, since the LSH algorithm is sensitive to the distances of the data points from the query point, and the Separation method make this distances further larger for "small" norms, hence, if we use "small" norms then the improvement in running time of this small norms is larger comparing to "larger" norms. However, this does not mean that in practice it is faster to use a "small" norm instead of the l_2 norm, since we do not know if the LSH algorithm for such "small" norms is as efficient as E^2LSH or not, as there is no practical version of such LSH algorithms.

B.1.1 Conclusions and important points

- The running time of the Near method is
  Time(Near method)=Time(E^2LSH)+Time(post-processing)
  =$O(1/\delta)O(dn^{\rho(c)}logn) + Time(post - processing)$, where $\rho(c) < \frac{1}{c}$.
- The running time of the Separation method is
  Time(Separation method)=$O(1/\delta)O((d+1)n^{\rho(c_2)}logn)$, where $\rho(c_2) < \frac{1}{c+\hat{\xi}}$.
- We conclude that the Separation method reduction improves the running time of the $PLDS$ using the E^2LSH algorithm, since the Separation method increases the distances of the non-cover points from the query, furthermore the Separation method does not need a post-processing.
- The ratio between the distances of most of the points and the maximum

threshold c_p is larger for ‘small‘ l_p norm. Therefore, the ratio of the improvement of the running time is better if we use ‘small‘ l_p norms.

- The E^2LSH is a randomized algorithm that find the near points with high probability (see 3.6). Note that the probability of finding a cover point when we use E^2LSH algorithm is the same guaranteed probability $1 - \delta$ by the algorithm for finding a near point within the search radius $R = r_{max}$. This is because in the new space the cover points are still inside the Ball $B(q, r_{max})$.

- For any constant δ the running time of the E^2LSH using the Separation method is theoretically always sub-linear, also in the cases that the running time of E^2LSH on the original data points is linear. More specifically, if there is no guarantee that most of the points are within distance $r_{max}(1 + \epsilon)$ (i.e. most of the points are originally inside the ball $B(q, r_{max})$), then the running time of the E^2LSH algorithm on the original space is $O(dnlogn)$ but in the new space it is less than $O((d + 1)n^{\frac{1}{c}}logn)$.

  This true since for *any* non-cover point p_j, its original distance must satisfy $dist_j = (1 + \gamma_j)r_j$ for some $\gamma_j > 0$, thus its new distance is $dist'_j = \sqrt{((1 + \gamma_j)r_j)^2 + r_{max}^2 - r_j^2}$ from eq.(6) therefore,
  $c_j = \frac{dist'_j}{r_{max}} = \sqrt{1 + (2\gamma_j + \gamma_j^2)\left(\frac{r_i}{r_{max}}\right)^2}$.
  Assume that most of the radii are around the average radius $\hat{r}$, we can estimate c above by $\sqrt{1 + (2\hat{\gamma} + \hat{\gamma}^2)\left(\frac{\hat{r}}{r_{max}}\right)^2}$. Where $\hat{\gamma}$ estimates the average γ of all the non-cover points, note that $c > 1$.

B.2 The Separation Method for the Unit Sphere

In the following we prove lemma 11 from subsection 5.2. We represent the lemma here

For a query q, applying the Separation method causes that any point p_k on the unit sphere which is originally cover the query lies inside the ball $B(q, r_{max})$, while if p_k is originally a non-cover point to q then it will lie outside $B(q, r_{max})$.

Proof. We consider the boundary point p_i (its radius equal its distance from the query) with the radius r_i see figure (4).

Assume that for any point p_k, $dist_k$ and $dist'_k$ are the distances from the query point before and after using the Separation method, respectively. Assume also $dist'_{ko}$ is the new distance of p_k from the origin. assume that p_k is any point such that its radius equal to the radius of the boundary point p_i i.e. $r_k = r_i$. If p_k is a boundary point then like the point p_i above and from equation (8) we have that $dist'_k = r^2_{max} = 1 + dist'^2_{ko} - 2dist'_{ko} \cdot cos(\alpha)$.

Note that p_k in the extreme case equal to the query, and in the other extreme case has distance 2 from the query. Thus, α_k belongs to the interval $[0, \pi]$, and the *cosine* function in this interval is a decreasing monotone function.

Assume first the extreme cases i.e. before applying the Separation method $p_k = q$ then after applying the Separation method we have from the case of the boundary point the distance of p_k is equal to y, thus included in the ball $B(q, r_{max})$. On the other hand, if p_k has originally distance 2 from the query then its new distance will be $2 + y$ thus not included in the ball $B(q, r_{max})$.

Most of the cases $dist_k$ satisfies $0 < dist_k < 2$ thus α_k belongs to $(0, \pi)$. Assume p_k is a cover point then, $dist_k < r_k$ and the angle between the data point and the query satisfies $\alpha_k < \alpha$ (α from fig. 4). Considering equation (8) will have that $dist'_k = 1 + dist'^2_{ko} - 2dist'_{ko} \cdot cos(\alpha_k)$. Note that $-2dist'_{ko} \cdot cos(\alpha_k) < -2dist'_{io} \cdot cos(\alpha)$ since $dist'_{ko} = 1 + y = dist'_{io}$ and $-cos(\alpha_k) < -cos(\alpha)$ in the given interval, thus $dist'_k < r_{max}$.

The proof for the case that p_k is non-cover point is in the same way.

■

B.3 Combining Multi-level and the Separation Method

In some cases we need to divide our original $PLDS$ problem into several subproblems and then solve every subproblem independently. This can be used to improve the running time when the maximum threshold is large compared with the inter point distances (see Multi-level 4.2), or to improve the running time if we use the approximate nearest neighbor algorithm and the ratio between the maximum threshold and the minimum threshold is large (see appendix B.4).

The Separation method was presented above (see 5) as a reduction from the $PLDS$ problem to the NNs problem both in theory and in practice. Hence, the closest point to the query after we run the Separation method must cover the query point if it is within distance r_{max} from the query, and every point included in the ball $B(q, r_{max})$ must cover the query point (see lemma 8 and cor. 9). Thus, it does not matter which algorithm we use for solving the NNs problem, either the nearest or the near neighbor algorithm; the Separation algorithm can solve the exact version of the $PLDS$ problem. Further, in both of these cases we do not need a post-processing step at all. Thus, the only reason that we divide the problem into subproblems is to optimize the running time of the algorithm. Therefore, we can divide the problem into levels without any restriction other than the running time.

B.3.1 How to divide

As we showed when we described the Direct Multi-level method, in the case that we used a near neighbor algorithm, we chose three different ways to divide the problem (see 4.2). We will try all of them here, hoping that it will improve the running time.

B.3.2 Comparing the Different Divisions Using the E^2LSH

The behavior of the Multi-level Separation method is similar to that of the Direct Multi-level method. Although they behave similarly, the improvement in the running time using the Separation method is greater than in the case of the Direct Multi-level method in all of the three divisions. On the other hand, the number of levels that optimize the running time of the Multi-level Separation method is slightly smaller than the number of levels in the case of the Direct Multi-level. The improvement in the power growth level widths division way for the Multi-level Separation method compared with the equal levels widths division way is not large. Similarly, the improvement using search growth level widths division way compared with the power Level widths division way is small or sometimes negligible.

B.3.3 The Running Time Using E^2LSH

We determine the number of levels by experimentally optimizing the running time. The analysis here is very similar to the analysis in the case of the Direct Multi-level method. If we assume as before that most of the points are at distance no less than c_iR_i from the query, where R_i is the radius of search in level i, then the estimation of c_i is a little different from the previous method, and it is as follows:

Previously we showed in (5.1.2) that for the Separation method, using just one level such that, $c_2 r_{max} = (1 + \epsilon + \hat{\xi}) r_{max}$, where c_2 is the ratio between the distances of most of the data points and the radius of search $R = r_{max}$. The estimation of c_2 is as follows, $c_2 = 1 + \epsilon + \hat{\xi} = \sqrt{(1+\epsilon)^2 + 1 - \left(\frac{\hat{r}}{r_{max}}\right)^2}$ (see eq.(10)), and $\hat{r}$ is the average threshold.

For the Multi-level case, we simply assume that R_i is the maximum threshold in level i, we estimate the average threshold in level i as $R_i - \Delta_i/2$, where Δ_i is the width of level i (see 4.2.1). Assume R_i is the radius of search for the algorithm in level i. Similar to eq.(3) we can estimate the c_i for level

i as

$$c_i = 1 + \epsilon + \hat{\xi} = \sqrt{(1+\epsilon)^2(\frac{r_{max}}{R_i})^2 + 1 - \left(\frac{R_i - \Delta_i/2}{R_i}\right)^2}. \qquad (12)$$

This estimation is only used for the general Separation method. The estimation of c_i for the Separation method on the unit sphere (subsection 5.2) is different. Similar to eq.(4) we can optimize the number of levels using the following estimated running time for k levels

$$O((d+1)\sum_{i=1}^{k} logn_i \cdot n_i^{\rho(c_i)}), \qquad (13)$$

where n_i as usual is the number of points at each level. For optimizing the Separation method on the unit sphere we used more accurate function of the running time for more details see appendix A. Note that as the number of levels increases, the running time of the Multi-level Separation method converges quickly to the running time of the Direct Multi-level method (this can be observed by simulating the running time and the number of levels). Thus, the number of levels that optimize the Separation method running time is less than the number of levels in the case of the Direct Multi-level method.

B.4 The Separation Method Using an $\epsilon - NNs$ Algorithm

As was mentioned in subsection 3.1, the problem of finding the nearest neighbor point exactly in high dimensions is conjectured to suffer from the curse of dimensionality. Thus, most of the algorithms were developed to solve the approximation version of the problem (the $\epsilon - NNs$). Here, we will analyze the Separation method using the approximate nearest algorithm in the l_1 and the l_2 norms.

First, we will present the case of solving the exact $PLDS$ problem, using an approximate nearest neighbor algorithm. Next, will present the case of solving the approximated problem ($\gamma - PLDS$).

B.4.1 Solving the Exact $PLDS$ Using an $\epsilon - NNs$

Assume that we are trying to solve the exact $PLDS$ problem (and not the $\gamma - PLDS$). To do so we have to ensure that this reduction "takes" the points at least as far as $r_{max} \cdot \epsilon$ from the sphere boundary, with radius r_{max} centered at the query in the new space $R_p^{(d+1)}$, where ($p = 1, 2$).

We are interested in points p_is that are within distance r_{max} from the query in the new space $R_p^{(d+1)}$, since these points cover the query in the original space (see Separation method in section 5).

Assume as before that p_j is the nearest neighbor to the query, $dist_j$, and $dist'_j$ represents the distances of p_j from q in the spaces R_p^d and $R_p^{(d+1)}$, respectively. If $\frac{r_{max}}{(1+\epsilon)} \leq dist'_j \leq r_{max}$, then according to the definition of the $\epsilon - NNs$, our approximated algorithm finds in the worst case point p_k in the range $r_{max} \leq p_k \leq (1+\epsilon)r_{max}$ from the query, as an approximate nearest point. In this case, we concluded that there is no cover point to the query q in the original space. However, this is a wrong conclusion; therefore, we should attempt to accomplish that the points that cover the query are within a distance less than $\frac{r_{max}}{1+\epsilon}$ in the original space, to ensure that the algorithm returns a cover point if it exists.

We now will check in which distance $dist_i$ point p_i should be in the original space R_p^d, so that its distance in the new space $R_p^{(d+1)}$ will be less than $\frac{r_{max}}{(1+\epsilon)}$. For the l_1 norm and for any point p_i,

$$\begin{aligned} dist'_i &= |dist_i| + |r_{max} - r_i|, \\ \frac{r_{max}}{1+\epsilon} &\geq |dist_i| + |r_{max} - r_i|. \end{aligned}$$

Thus, the original distance should satisfy

$$dist_i \leq r_i - \epsilon r_{max}. \tag{14}$$

For the l_2 norm and for any point p_i,

$$\begin{aligned} dist'_i &= \sqrt{(dist_i)^2 + r_{max}^2 - r_i^2}, \\ \frac{r_{max}}{(1+\epsilon)} &\geq \sqrt{(dist_i)^2 + r_{max}^2 - r_i^2}. \end{aligned}$$

Thus the original distance should satisfy

$$dist_i \leq \sqrt{r_i^2 - r_{max}^2 \left(\frac{2\epsilon + \epsilon^2}{(1+\epsilon)^2} \right)}. \tag{15}$$

The inequalities 14 and 15 above show that for solving the exact version of the $PLDS$ problem, the distances should satisfy such a restriction. We note that this restriction is less robust if we use the l_2 norm. We will call the distances in the inequalities above (14 and 15) the critical distances. Note that the ratio between r_{max} and r_i play a major rule in determining the critical distance from the query. Unfortunately, we noticed that for practical values of ϵ even for reasonable ratios between r_{max} and r_i, the critical distance for the l_2 norm is small relative to its threshold r_i. Thus, we can get unsatisfactory results using the l_2 norm in the sense that we will miss some cover points. In order to make this problem less delicate, we need to choose a relatively small ϵ, which could has bad performances in practice. A better way to solve this problem is to use the Multi-level Separation method (see subsection 5.3 and appendix B.3).

B.4.2 Solving the $\gamma - PLDS$ Using an $\epsilon - NNs$

We will now analyze the case of solving the approximated problem ($\gamma - PLDS$). Assume that we allow the approximation factor γ for solving the

$PLDS$ problem. We need to compute the suitable approximation factor ϵ for the $\epsilon - NNs$ algorithm that gives us a suitable γ.

Similar to the analysis above, if the nearest point p_j distance in the new space satisfies $\frac{r_{max}}{(1+\epsilon)} \leq dist'_j \leq r_{max}$, then according to the definition of $\epsilon - NNs$, our approximated algorithm finds in the worst case point p_k that satisfies $r_{max} \leq p_k \leq (1+\epsilon)r_{max}$ as an approximate nearest point. We need to ensure that such a point p_k has a distance that is at most $1+\gamma$ fraction of its radius r_j, i.e. for the l_1 norm and for *any* point p_k that has $dist'_k = (1+\epsilon)r_{max}$,

$$\begin{aligned} dist'_k &= |dist_k| + |r_{max} - r_k|, \\ r_{max} \cdot (1+\epsilon) &= |dist_k| + |r_{max} - r_k|. \end{aligned}$$

Thus, the original distance should satisfy

$$dist_k = \epsilon r_{max} + r_k.$$

Thus,

$$1 + \gamma \geq \frac{dist_k}{r_k} = \frac{\epsilon \cdot r_{max} + r_k}{r_k},$$

and the relation between γ and ϵ will be

$$\epsilon = \gamma \frac{r_k}{r_{max}} \geq \gamma \frac{r_{min}}{r_{max}}.$$

To ensure that for points with r_{min} it satisfies the request, we should take

$$\epsilon \leq \gamma \frac{r_{min}}{r_{max}}.$$

For the l_2 norm and for *any* point p_k that has $dist'_k = (1+\epsilon)r_{max}$,

$$dist'_k = \sqrt{dist_k^2 + (r_{max}^2 - r_k^2)},$$

$$(1+\epsilon)r_{max} = \sqrt{dist_k^2 + (r_{max}^2 - r_k^2)}.$$

Thus,

$$dist_k = \sqrt{((1+\epsilon)r_{max})^2 - (r_{max}^2 - r_k^2)}.$$

The ratio

$$(1+\gamma) = \frac{dist_k}{r_k} = \sqrt{\frac{((1+\epsilon)r_{max})^2 - (r_{max}^2 - r_k^2)}{r_k^2}}.$$

Thus $1+\epsilon$ satisfies

$$(1+\epsilon) = \sqrt{\frac{(1+\gamma)^2 + (\frac{r_{max}}{r_k} - 1)^2}{(\frac{r_{max}}{r_k})^2}},$$

also for a smaller $1+\epsilon$ value it will work; thus taking

$$1+\epsilon = \sqrt{(1+\gamma)^2(\frac{r_k}{r_{max}})^2},$$

is sufficient. To ensure that for points with r_{min} it will satisfy the request, we should take

$$\epsilon \le \gamma\frac{r_{min}}{r_{max}}.$$

Hence, both in the case of the l_1 norm, and the l_2 norm if we take the approximation factor to the $\epsilon - NNs$ algorithm to be $\epsilon = \gamma\frac{r_{min}}{r_{max}}$, then we will satisfy the request that if there exists a point p_j such that $q \in B(p_j, r_j(1+\gamma))$, then the approximated nearest algorithm will return p_k such that $q \in B(p_k, r_k(1+\gamma))$.

Note that in the case of the l_2 norm we can take a larger ϵ value that satisfies the goal (we can take $1+\epsilon = \sqrt{\frac{(1+\gamma)^2+(\frac{r_{max}}{r_k}-1)^2}{(\frac{r_{max}}{r_k})^2}}$).

Thus, for solving the $\gamma - PLDS$ problem, the l_2 norm has a larger ϵ value for the $\epsilon - NNs$ than the l_1 norm.

C Intersection Method

C.1 The Algorithm

- Pre-processing the points of the data structures

```
1- Prepare for every point $p_i$ in the data set one data structure; call it
   $list_i$.

2- For i=1 until n

     3- For j=1 until n
     4- If $(i \neq j)$ then
          a- Find $dist_{ij}$ */distance between the points $p_i, p_j$*/
          b- If $(dist_{ij} \leq 2 \cdot r_j)$ */(if $p_i$ will be the nearest point, then every
             point that covers the query with $r_j$ must also cover $p_i$ with
             $2r_j$) */
             then save the index $(j)$ in $list_i$.
     5- End if
     6- End for

   7-End for
```

- Query Processing

```
1- Assume that $q$ is the query point. Run a nearest neighbor algorithm
   and assume that $p_m$ (for some $m$) is the nearest neighbor found by the
   algorithm.
```

2- If $d(q, p_m) > r_{max}$
then return '*No* cover points in the data set'

else
If there is any point whose index is stored in $list_m$ that covers the query point, then return it as the answer.
If no point from $list_m$ covers the query return '*No* cover points in the data set.'

C.2 Analysis When Using the E^2LSH

Practical approach: we have available a near neighbor algorithm; how can we use it as a nearest neighbor algorithm? We will not use a real nearest algorithm but instead, we plan to use the near neighbor algorithm E^2LSH (section 3.6) because of its efficient performance. We can find the nearest point to the query by searching the near points within a reasonable distance R returned by the $R - NN$ algorithm. The only problem remaining is to determine R, the suitable search radius of the algorithm. In order to make the Intersection method using E^2LSH efficient, the nearest neighbor point distance to the query should be significantly smaller than the maximum threshold r_{max}. Therefore, in most of the cases we do not need to run the algorithm E^2LSH with a large search radius R (that equals r_{max}) just for the purpose of finding the nearest neighbor point.

Thus, the Intersection method, using the E^2LSH algorithm as a "nearest" neighbor algorithm, gives good performances for situations where the nearest neighbor point is meaningful, i.e. for clustered data points or when the underlying dimensionality of the data points is much lower than the actual dimensionality (for more details see [5]). We will focus on clustered data points in our experiments, such that we require that with a high probability the query point falls within one of the data clusters. This situation is perfectly realized in the classification problem, where data naturally falls into discrete classes or clusters in some potentially high dimensional feature space. It is one of the few realistic situations where the nearest neighbor point is considered to be meaningful [5].

Thus, in real applications involving classifications, the data points are usually clustered or are implicitly in low dimensionality.

Finding the suitable search radius R for the E^2LSH algorithm is an easy task, since in high dimension the distance of the nearest neighbor can usually be predicted. In most cases we succeed in finding the nearest neighbor the first time, but if we fail we can use a binary search on R in the interval of thresholds values $[r_{min}, r_{max}]$ until we find the nearest neighbor point. It is obvious that the probability of finding a cover point, if it exists using E^2LSH under the Intersection method, is at least, as the probability of finding a near point using E^2LSH algorithm $(1-\delta)$.

C.2.1 The Running Time Using the E^2LSH

Assume that S_i is the size of $list_i$ for any point p_i. The worst case running time is less than $O(log(r_{max}-r_{min}) \cdot Time(E^2LSH)) + d \cdot S_{max}$, where S_{max} is the size of the maximum list .

The log term appears since in the worst case we will fail to find the nearest neighbor, and we will therefore need to change the search radius, using a binary search until we reach the maximum radius r_{max}.

D Similarity with Virtual Levels

D.1 The Virtual Levels

Here, we show how we can apply the Similarity method in practice. First, we will discretize the thresholds interval values to several sub intervals, then we will virtually classify the data set points with respect to the additional dimension by their thresholds. Consequently, we will have virtual levels, for which every one of them has its maximum radius.

D.1.1 How to apply the method.

We analyze the method for the l_2 norm only, since we are interested in the l_2 norm for our applications.

First, we will discretize the interval $[r_{min}, r_{max}]$ where r_{min} and r_{max} are the minimum and the maximum radii in the data set, respectively. Assuming that k is the number of discrete intervals, we will have the values $R_1, R_2, ..., R_k$ as the maximum discrete radii, where the discretization step $\Delta = (r_{max} - r_{min})/k$ and $R_i = r_{min} + i \cdot \Delta$.

Now, for every value of R_i we produce a query point q_i such that in its additional dimension we substitute the value $\alpha(R_i) \cdot R_i$, where $\alpha(R_i)$ will be represented later. Assume the original query $q = \{q^{(1)}, q^{(2)}, \cdots, q^{(d)}\}$ then query q_i satisfies,

$$q_i = \{q^{(1)}, q^{(2)}, \cdots, q^{(d)}, \alpha(R_i) \cdot R_i\}. \tag{16}$$

On the other hand, we need to find the optimal value for k, which also will be discussed later.

We consider any point p_j such that its radius r_j satisfies $R_{i-1} < r_j <= R_i$ as belonging to the virtual level numbered i, we substitute the value $\alpha(R_i) \cdot r_j$ in its additional dimension. If $p_j = \{p^{(1)}, p^{(2)}, \cdots, p^{(d)}\}$ in the original space then in the new space p_j satisfies,

$$p_j = \{p^{(1)}, p^{(2)}, \cdots, p^{(d)}, \alpha(R_i) \cdot r_j\}. \tag{17}$$

We can show that if the point p_j is a cover point then it is included in the ball with radius R_i centered at q_i. Furthermore, a negligible amount of non-cover points with radii belong to the interval $[R_{i-1}, R_i]$ will be included in the ball $B(q_i, R_i)$.

Lemma 13 *Assume we take $\alpha(R_i) = \sqrt{(2R_i - \Delta)/\Delta}$ for each data point p_j included in virtual level numbered i i.e. whose radius $r_j \in (R_{i-1}, R_i]$. We claim that each point belongs to virtual level i, that covers the original query q in the original space R_2^d is included within the ball with radius R_i centered in the query q_i in the new space $R_2^{(d+1)}$.*

Proof. Assume that $dist_j$ is the distance of the point p_j in the original space from the query point, and $dist'_j$ is the distance of p_j from the query q_i in the new space.

In the worst case, p_j is a level boundary point; we define a level boundary point as a boundary point whose radius is smaller or equal (larger or equal) to the minimum (maximum) threshold in the level. This means that its radius $r_j = R_{i-1} = R_i - \Delta$ (or $r_j = R_i$), and its distance from the query in the original space R_2^d equals its radius. (Note that if $r_j = R_{i-1}$, then p_j does not belong to level i but for our analysis we assume that it is the case). We need to satisfy for each cover point in the original space R_2^d, which included in the current level i in the new space $R_2^{(d+1)}$ the following.

If p_j covers q in the original space $R_2^d \Longrightarrow p_j \in B(q_i, R_i)$ in the new space $R_2^{(d+1)}$, then

$$dist'_j = \sqrt{\sum_{k=1}^{d+1} \mid p_j^{(k)} - q_i^{(k)} \mid^2} \leq R_i,$$

from eq.(16), and eq.(17)

$$dist'_j = \sqrt{dist_j^2 + (\alpha(R_i) \cdot R_i - \alpha(R_i) \cdot r_j)^2} \leq R_i.$$

In the worst case p_j is a level boundary point; the interesting case is when $r_j = R_i - \Delta$,

$$dist'_j = \sqrt{(R_i - \Delta)^2 + (\alpha(R_i)(R_i - r_j))^2} \leq R_i,$$

$$\sqrt{(R_i - \Delta)^2 + \alpha^2\Delta^2} \le R_i,$$
$$(R_i - \Delta)^2 + \alpha^2\Delta^2 \le R_i^2,$$
$$\alpha(R_i) \le \sqrt{(2R_i - \Delta)/\Delta}.$$

Therefore, if we assume equality in the last inequality, we will get that level boundary points fall on the boundary of R_i ■

We can show in the same way as in lemma (13) above, that any point p_j such that $r_j \in (R_i, R_{i+1}]$, i.e. p_j belongs to the next virtual level $(i+1)$, included in the current ball $B(q_i, R_i)$, if and only if, p_j is a cover point for the original query.

Thus, "bad" points that belong to a higher virtual levels cannot be included in the current ball $B(q_i, R_i)$.

We showed above that every point in the current virtual level i, which covers the query in the original problem, must lie inside the ball with radius R_i centered at q_i.

In the next lemma we show that if we make Δ significantly small compared with the minimum threshold r_{min}, then a negligible amount of points that belong to the current level and do not cover the original query lie inside the ball $B(q_i, R_i)$. More precisely, if p_j included in the ball $B(q_i, R_i)$ then the ratio between its distance and its radius is at most $\sqrt{1 + \frac{R_i \cdot \Delta}{2(R_i - \Delta/2)^2}}$.

Lemma 14 *Any non-cover point p_j that belongs to the current level i, i.e. $r_j \in (R_{i-1}, R_i]$, is not included in the ball $B(q_i, R_i)$ in the new space $R_2^{(d+1)}$, If the ratio $\frac{dist_j}{r_j}$ satisfies $\frac{dist_j}{r_j} \ge \sqrt{1 + \frac{R_i \cdot \Delta}{2(R_i - \Delta/2)^2}}$, where $dist_j$ is the distance of p_j from the original query in the original space R_2^d.*

Proof. Assume p_j is not a cover point, such that $r_j = R_i - \tilde{\Delta}$, where $0 < \tilde{\Delta} \le \Delta$, and $dist_j = R_i + \gamma$ (such that $-\tilde{\Delta} < \gamma$). Consider that $\tilde{\Delta} = \beta \cdot \Delta$, where $0 < \beta \le 1$.

We will check when the point p_j is on the boundary of $B(q_i, R_i)$ in the worst case.

$$dist'_j = \sqrt{\sum_{k=1}^{d+1} \mid p_j^{(k)} - q_i^{(k)} \mid^2} = R_i,$$

from eq.(16),and eq.(17)

$$dist'_j = \sqrt{dist_j^2 + (\alpha(R_i)R_i - \alpha(R_i)r_j)^2},$$
$$dist'_j = \sqrt{(R_i + \gamma)^2 + (\alpha(R_i)\beta\Delta)^2} = R_i.$$

From lemma(13) $\alpha(R_i) = \sqrt{(2R_i - \Delta)/\Delta}$, thus

$$(R_i + \gamma)^2 + (\frac{2R_i - \Delta}{\Delta})\beta^2\Delta^2 = R_i^2.$$

This is true if γ satisfies

$$\gamma_{1,2} = -R_i \pm \sqrt{R_i^2 - 2\beta R_i\tilde{\Delta} + \tilde{\Delta}^2}.$$

The worst case for the parabola under the square root is when $\beta = \frac{1}{2}$, then,

$$\gamma_{1,2} = -R_i \pm \sqrt{(R_i - \tilde{\Delta})^2 \quad \underbrace{+R_i\tilde{\Delta}}_{\text{"the bad term"}}}.$$

The ratio between the distance and the radius in the worst case ($\beta = \frac{1}{2}$) is

$$\frac{dist_j}{r_j} = \frac{R_i + \gamma}{R_i - \tilde{\Delta}} = \pm\sqrt{1 + \frac{R_i \cdot \Delta}{2(R_i - \Delta/2)^2}},$$

which is logical just for a positive term, thus

$$\frac{dist_j}{r_j} = \sqrt{1 + \frac{R_i \cdot \Delta}{2(R_i - \Delta/2)^2}}.$$

■

Now we need to show that “bad“ points from other virtual levels cannot be included in the ball $B(q_i, R_i)$ in the current virtual level i.

Lemma 15 *If p_j does not cover the original query in the original space R_2^d, and $r_j \in (R_{k-1}, R_k]$, i.e. p_j belongs to the virtual level number k, then p_j is not included in the ball $B(q_i, R_i)$ that corresponding to virtual level i, in the new space $R_2^{(d+1)}$, where $i \neq k$.*

Proof. Assume that $dist'_j$ is the distance of the point p_j from the query q_i in the new space, and $dist_j$ is as usual the original distance of p_j from the original query.

Since p_j belongs to the kth virtual level, then $r_j = R_k - \tilde{\Delta}$, where $\tilde{\Delta} = \beta \cdot \Delta$, and $0 < \beta \leq 1$.

Assume without loss of generality that $i > k$. Then we need to show that

$$dist'_j = \sqrt{\sum_{t=1}^{d+1} \mid p_j^{(t)} - q_i^{(t)} \mid^2} > R_i,$$

from eq.(16), and eq.(17)

$$dist'_j = \sqrt{dist_j^2 + (\alpha(R_i)R_i - \alpha(R_k)r_j)^2} > R_i,$$

Since $\alpha(R_i) > \alpha(R_k)$ and $R_i > r_j$, then

$$\sqrt{dist_j^2 + (\alpha(R_i)R_i - \alpha(R_k)r_j)^2} > \sqrt{dist_j^2 + (\alpha(R_i)R_i - \alpha(R_i)r_j)^2}.$$

Note that $dist_j > r_j = R_k - \tilde{\Delta}$; hence, we have

$$dist'_j > \sqrt{(R_k - \beta\Delta)^2 + (\alpha(R_i)(R_i - (R_k - \beta\Delta)))^2}.$$

Since $i - k \geq 1$ then in the worst case $R_k = R_{i-1} = R_i - \Delta$, i.e. level k in the worst case is just level $(i-1)$. Thus,

$$\sqrt{(R_i + (-\beta - 1)\Delta)^2 + (\alpha(R_i)(\beta + 1)\Delta)^2} \geq R_i.$$

Finally, we find that $\alpha(R_i)$ should satisfy

$$\alpha(R_i) \geq \sqrt{\frac{2R_i - (\beta+1)\Delta}{(\beta+1)\Delta}}.$$

Note that for the given values of β,

$$\sqrt{\frac{2R_i - (\beta+1)\Delta}{(\beta+1)\Delta}} \leq \sqrt{\frac{2R_i - \Delta}{\Delta}}.$$

We substituted for $\alpha(R_i)$ the value $\sqrt{\frac{2R_i-\Delta}{\Delta}}$ (see lemma 13). Therefore, we satisfied the request above that bad points from other virtual levels are not included in the current virtual level i. In this way, we can guarantee that any "bad" point included in virtual level k cannot be include in $B(q_i, R_i)$ the ball in the current virtual level i. ■

D.1.2 The Virtual Levels Algorithm

a- **Virtual Levels Using a Nearest Neighbor Algorithm**
Assume that we use a nearest neighbor algorithm for solving the similarity problem in the new space $R_2^{(d+1)}$. The algorithm returns the closest point as a cover point in any level i, if the distance of the closest point in the new space $R_2^{(d+1)}$ is smaller than R_i, where as above, R_i is the maximum threshold in level i.

In this scenario the problem that we solve is the $\gamma - PLDS$ problem (see definition 2 section 3.3), and not the exact $PLDS$. Assume that the point p_j belongs to virtual level i, such that its distance in the original space R_2^d satisfies $r_j \leq dist_j \leq \left(\sqrt{1 + \frac{R_i \cdot \Delta}{2(R_i - \Delta/2)^2}}\right) \cdot r_j$. This point is not a cover point but it can be included in the ball $B(q_i, R_i)$ (see lemma 14). Thus, in some cases it can be chosen as the closest neighbor point.

Assume $1 + \gamma = \left(\sqrt{1 + \frac{R_1 \cdot \Delta}{2(R_1 - \Delta/2)^2}}\right)$, and that we have k virtual levels and $R_1 = r_{min} + \Delta$. Note that γ depend on r_{min}, in other words, the approximation factor γ is determined by the value of r_{min}. In the worst case $r_{min} = 0$, then $1 + \gamma = \sqrt{1 + \frac{\Delta^2}{\Delta^2/2}} = \sqrt{3}$, regardless of the value of Δ, i.e. regardless of the number of Virtual Levels used. If $r_{min} > 0$, then the ratio between Δ and r_{min} determines the value of γ.

Any point p_j that satisfies $dist_j \leq (1+\gamma) r_j$ can be chosen as the nearest neighbor point. In this case the solution is for the $\gamma - PLDS$ problem and not for the exact $PLDS$ problem, similar to the case of the Direct Multi-level method. It can be easily shown that for a large enough r_{min}, and for the same number of levels, the γ value in the case of the Virtual Levels method are much smaller than the γ value in the case of the Direct Multi-level method. Thus, in this case the virtual levels method solves $\gamma - PLDS$ with greater accuracy.

The Algorithm

- <u>Pre-processing</u>

1- Choose k, the number of levels that gives the required approximation factor γ (see lemma 14).

2- Create the Virtual Levels as follows. Compute $(R_1, ..., R_k)$, where $R_i = r_{min} + i \cdot \Delta$ and $\Delta = \frac{r_{max} - r_{min}}{k}$. Add for every data point a new dimension such that, for any point p_j that has a threshold $r_j \in (R_{i-1}, R_i]$, we substitute in the new dimension the value

$\alpha(R_i) \cdot r_j$ (see eq. 17), such that $\alpha(R_i) = \sqrt{\frac{2R_i - \Delta}{\Delta}}$ (see lemma 13). In this case we say that the point p_j belongs to Virtual Level i.

- Query processing

1- Build k queries $q_1, q_2, \cdots, q_k$ using the original query and an additional dimension, where the value $\alpha(R_i) \cdot R_i$ is substituted for the additional dimension for every query q_i (see eq. 16), such that $(1 \leq i \leq k)$.

2- For every query from step (1), run a nearest neighbor algorithm. If the distance of the closest point from the query q_i in the new space is smaller or equal to R_i, return it as a cover point. Otherwise, if all the queries have negative answers, then return '*NO* cover point for the original query'.

b- **Virtual Levels Using a Near Neighbor Algorithm**

Assume that the near neighbor algorithm reports all the points within radius R, for any given R. Using the $R - NN$ algorithm solves the exact $PLDS$ problem.

The Algorithm

- Pre-processing

1- Choose k, the number of levels that gives the required Δ for optimizing the near neighbor algorithm running time.

2- The same as step (2), in the pre-processing of the nearest neighbor algorithm above.

- Query processing

1- Build k queries $q_1, q_2, \cdots, q_k$ using the original query and an additional dimension, where the value $\alpha(R_i) \cdot R_i$ is substituted for the additional dimension for every query q_i , such that $(1 \leq i \leq k)$.

2- For query number i from step (1) run a $R - NN$ algorithm on a search radius equal to R_i. From the points that the algorithm returns if any cover the original query, then return it as a cover point. If we fail to find a cover point in all the Virtual Levels (i.e. for all the k queries), then return 'NO cover point for the original query'.

Note: As we mentioned in the case of the Direct Multi-level method (sec. 4), and the case of the Separation method (sec. 5), using the Virtual Levels method with a post-processing step, based on a $R-NN$ algorithm solves the exact $PLDS$ problem.

Thus, as we did for the previous methods, we can choose the optimal number of levels as the value that optimize the running time function of the $R - NN$ algorithm that we use.

D.2 Virtual Levels Using the E^2LSH

As we showed in the algorithm above, each data point, say p_j, has a new dimension, which we substitute for it the term $\alpha(R_i) \cdot r_j$. We also showed that every query point q_i corresponding to level i has the value $\alpha(R_i) \cdot R_i$ in its new dimension ,see eq.(16),and eq.(17).

The running time of the E^2LSH algorithm depends on the distances of the data points from the query point. Assume p_j is any data point that belongs to level i. Since E^2LSH is sensitive to the distances, we found it very useful to add to the value of the new dimension $(d + 1)$ for the point p_j, in addition to the term above, the term $(i \cdot Divide)$. The term $(Divide)$ is some constant that is related to the intermediate distances between the points. It is obvious that the term $(i \cdot Divide)$ should be also added to the value of the new dimension of the query q_i for every $(1 \leq i \leq k)$. Doing this improves the running time of the Virtual Levels in the case where we use the near neighbor algorithm E^2LSH for the similarity.

The running time of the Virtual Levels using the constant ($Divide$) in this case is somehow similar to the running time of the Multi-level Separation method. Therefore, as we showed using the Multi-level Separation method (sec. 5.3), the optimal number of levels using the Separation method is almost the same as using the Direct Multi-level method. The optimal number of levels for the Virtual Levels can be determined in the same way. We can use the optimal value of the Multi-level Separation method as an estimation to the optimal value of the number of levels for the Virtual Levels method.

E Comparing the Different Methods

This section compares the different methods with one another, for solving the $PLDS$ problem, and determining the running time and the storage space required.

Each method previously mentioned has its unique main idea that it used to solve the problem; the Direct Multi-level method relies mainly on dividing the data set points. The Separation method mainly reduces the $PLDS$ problem to the NNs problem by changing the distances. In contrast, the Intersection method relies heavily on the mean inter-point distance compared with the radii; the radii used should be at least twice as small as the mean inter-point distance. The Similarity with Virtual Levels method works by producing many queries instead of the original query, then classifies the data points to virtual levels by increasing their distances.

If a NNs algorithm is used, the methods that are used to solve the exact version of the $PLDS$ problem are the Separation method and the Intersection method. The Direct Multi-level and the Virtual Levels methods solve the $\gamma - PLDS$ problem and not the exact $PLDS$, but there is a difference between the approximation factor γ in these two methods.

If a $R - NN$ algorithm is used, then all of the methods will solve the exact $PLDS$ problem. But the Separation method is the only method that does not need a post-processing step.

E.1 Comparing the Running Time

To compare the performances of the methods using the E^2LSH algorithm see section 8. In general, it is difficult to compare the running time of these methods, since the running time is different for different NNs algorithms. It is possible that we would prefer a specific method for a particular NNs algorithm and another method for other algorithm. This depends on the properties and the performance of the algorithm using the specific method. Sometimes it depends also on the data set distances and the radii distributions. However, generally methods that increase the distances of the data points from the query points relative to the radius search (such as the Separation method) are better for the near neighbor algorithms, and methods that increase the distances of the approximate nearest neighbor points from the nearest neighbor point (such as the Direct Multi-level method) are better for the nearest neighbor algorithms.

E.2 Comparing the Storage Space Required

Here we determine the space requirements for the methods without considering the NNs algorithm that we use. We assume that the same NNs algorithm is used as a black box for all the different methods.

The additional space that we need for the Separation method is $O(n)$, since we need for every point (and for the query) just one additional dimension. In addition, the Similarity with Virtual Levels method needs $O(n)$ additional space for the same reason. Assume k is the number of virtual levels, we also need space to the k new queries but we assume that the number of levels is negligible compared with n.

In the case of the Direct Multi-level method we do not need any additional space, just $O(1)$ space for maintaining information about the levels.

For the Intersection method, if we assume that the average number of points that is saved in the data structure lists is $O(m)$, then the additional space that we need is $O(mn)$ for all the data points. In the worst case $m = n$, but the worst case is not practical at all, since in this case the algorithm has a worse running time than the naive search algorithm, and we do not have any interest in running the Intersection method in such cases.

E.3 Comparing the Methods for Dynamic Databases

Dynamic databases are databases that support *Inserting/Deleting* points. Here we discuss two questions "Is it possible to support a dynamic database by our methods? ", and "What are the 'costs' of supporting inserting/deleting using the different methods mentioned above? "

Our discussion is on the level of the methods themselves, and assuming that they are using as a black box the same NNs algorithm, we will not discuss the inserting/deleting for the black box NNs algorithm itself.

Theoretically, all of the methods above can support inserting/deleting points. But they differ by the cost of these operations. Inserting a new point using the Direct Multi-level method is performed easily by entering the new point at its suitable level corresponding to its radius, if its radius is within the interval $[r_{min}, r_{max}]$. If its radius is not belonging to the interval $[r_{min}, r_{max}]$, then we have to build a new level for the new point and change r_{max} or r_{min}, respectively. A Similar analysis can be done to the Virtual Levels method, but in addition, if we build a new virtual level, then we need to create also a new query for this virtual level.

The way we insert a point in the case of the Multi-level Separation method (or the one level Separation method) is also similar to the way we insert a new point using the Direct Multi-level method. If the one level Separation method is used and the new point radius is bigger than r_{max}, then we have also to change the value of the additional dimension $(d+1)$, for all the points in the database to a new value related to the new maximum radius. This change may cost $O(n)$ operations.

In the case of deleting points from the database, it may need $O(n)$ operations in the three methods above, since we need to search the database (at least one level) for that point. On the other hand, if the points have indices that can be used as ID numbers, then the deletion may cost just $O(1)$ operations. Deleting may cause the number of levels to collapse, since one level may become empty after the deletion. Similarly to the case of inserting, sometimes there is a need to change the additional dimension of the one level Separation method. This occurs when we delete the last point, which has a radius that equals to the maximum radius.

For inserting/deleting operations analysis using the Intersection method, see subsection E.3.1.

E.3.1 Inserting and Deleting for the Intersection Method

Does the Intersection method support inserting new points and deleting points from the data set?

Assume that the number of points that are stored in every data structure point list is small compared with n, the number of data points. consequently, if most the data points have radii that are sufficiently small compared with the mean inter-point distance, then we can perform insertion of a new data point p_{n+1} in $O(n)$ time. The reason is that we have to run all over the data points and check whether the new point ball with twice its radius includes other data points, and store it in every data structure list whose corresponding data point included.

Similarly, if we want to delete some data set point p_i, it will take $O(n)$ time for the analogous reason that we have to run over all the points and check their lists to delete the point p_i if it is found there.

If in our application the insert operations are significantly greater than the delete operations, then we can just mark the point that we need to delete with some special flag, and then simply ignore it during the running application.

F Related Work

Some researchers in computer vision who had faced a similar problem in their databases regarding recognition, approached the problem essentially by reducing the number of queries. Given the new image for recognition, they do not build a query from every possible patch of the new image, but rather run an algorithm that computes a small number of feature points from the new image. In this way they drastically reduce the number of queries to the image database (see [12, 20]).

We chose to find an algorithm that reduces the running time of each query such that the overall running time of all the queries is reduced by a large amount. Our solution is based on the near/nearest search problem.

Indexing similarity in high dimensions is an active research field. Thus, we can find numerous works for the nearest neighbor search and for the near neighbor search, some of which rely on partitioning the space, for example the KD tree, the SR tree, the X tree, and the TV tree [19] (for a survey see [7]). Recently, approaches were developed based on the *locally sensitive hash functions* such as [16, 13, 11] (see subsection 3.4 of the background). Although there is a vast amount of literature on algorithms for the NNS (ε-NNS), there are only a few studies related to the $PLDS$ problem. One recent example is a study of multimedia identification, which was applied for Audio Fingerprinting by Goldstein et al, [14]. The motivation for this work was to find matches to small parts of songs, which they called audio fingerprints, from other fingerprints stored in the database. Similar to our problem, in their database every fingerprint has a threshold.

These thresholds define the d-dimensional spheres centered at the fingerprints, which they called hyper-spheres. They approximate the hyper-spheres by circumscribing or non-circumscribing hyper-rectangles.

They use non-circumscribing hyper-rectangles if some false negative results are allowed. For efficiently solving the rectangles search problem, they use bit vector indices. Let us assume that we give every database point an integer number as an ID. A bit vector for some dimension is a vector that corresponds every database point ID to a bit of binary value, whose value determines whether the projection of the rectangle in that dimension intersects the query or not. More precisely, each dimension is partitioned into several intervals, and for every interval a corresponding bit vector is defined. Each bit of such a vector corresponds exactly to one rectangle from the data set; thus, the vector length is equal to the number of data points (n). Further, each bit vector determines which rectangle projections overlap its corresponding interval and which do not by fixing zero/one values for every bit.

This technique partitions the queries per dimension, such that each partition contains all the IDs of the hyper-rectangles that overlap the query partition description in that dimension. The bit vector indices are just a way of compressing those rectangle ID lists, so that the lists can be intersected efficiently, considering that AND and OR are very fast operations in the computer.

The main advantage of using our methods comparing to this technique is that the running time of our methods based on the E^2LSH algorithm solve the $PLDS$ problem in typically sub-linear time on the data points, compared with the linear dependence of this technique on the data points. The running time using this technique is still $O(dn)$, the same order as the naive search.

Our methods are more generic and general since they are reductions from the $PLDS$ (or $\gamma - PLDS$) problem to the nearest or near neighbor problem; thus, they can be used for any near/nearest algorithm. Any improvement in the algorithms used in the field of the nearest neighbor will cause an improvement in the performances of our methods. On the other hand, the method of Goldstein et al, is a specific method that is based on the speed of the binary operations.

We also noticed that for solving the problem efficiently, Goldstein et al's method needs to be utilized and more information is needed about the data points distribution, the query distribution and the thresholds distribution. More specifically, this method needs to observe the projected distribution of the data points and thresholds in every dimension, and sometimes it also needs to have information about the query distribution to achieve an efficient partitioning for each dimension. On the other hand, our methods need less information about the data points; the only necessary step in our case that needed for the Multi-level methods is to estimate $(1 + \epsilon)$, the ratio between the distances of most of the points and the maximum threshold.

In contrast to most of the algorithms that suffer from the curse of dimensionality our methods relying on the E^2LSH algorithm improve their running time as the dimensionality increases, we call such a phenomena 'the benefit of dimensionality'. The improvement of the running time is due to the fact that when the dimensionality increases the data points become more equidistant i.e. close points to the query increases their distances[5], this improve our methods running time using the E^2LSH in a noticeable manner. Thus, our methods are suitable for data points with extremely high dimensions. On the other hand the increase of dimensionality makes the performance of the method of Goldstein et al worse. As we mentioned above, the method of Goldstein et al was implemented to audio fingerprinting system that contain $\sim 240,000$ data points of 64 dimensions. The thresholds were taken as a fixed fraction of the mean distance of the data points and the fractions ranged from $0.3 - 0.5$ of the mean distance (the average around 0.4) (for more details see [6]). The improvement compared with the naive search where ~ 50 times better. The comparing was done just for 14 dimensions, since they are the most informative for the audio database.

From our simulations on points uniformly distributed on the unit sphere with mean distance equal $\sqrt{2}$ with 64 dimensions, the improvement in running time for the general clustering method was ~ 25 better, Note that the dimensionality of the data points affect our methods performance, thus we repeat the experiment for points distributed on the unit sphere with 500 dimensions and get improvement in the running time of ~ 70 better than the naive search. Further, due to the sub-linear running time function of the E^2LSH algorithm, the improvement of our methods increases with the number of points n. Therefore, for large enough n our methods have better running time also for the audio database with 64 dimensions.

The increase of dimensionality and the number of data points improves the performances of our method but, on the other hand it, makes problems for Goldstein et al method. It is required in their method to compare all the dimensions if there are other database points that do not have special informative dimensions. This causes that they need a significantly larger space to make a finer partition to the different dimensions. Furthermore, they need to do the AND and the OR binary operations for all of the dimensions, we are not sure that running the binary operations for points in feature space with very large number of dimensions (say 1024 dimensions or larger) has the same efficiency as above, because of the limitation of the size of the cash memory and the need to do page exchanges with the main memory. On the other hand E^2LSH which we rely on, still works efficiently on feature spaces with very high dimensions.

Note that the running time improvement and space required of their method are very related to the values of the points thresholds. For larger thresholds this method needs significantly more space to build more bit vectors to a finer partitioning for maintaining the running time efficiency. In a similar manner the E^2LSH algorithm running time and space requirements are dependent on the values of the data points thresholds.

The advantage of Goldstein et al's method comparing to our methods if we do not consider the intersection method is that this method improves the running time for both the $PLDS$ problem and the $PLES$ problem, where the

$PLES$ problem is the same as the $PLDS$ problem if we consider equal radii for all the points. The Direct Multi-level method, the Separation method, and the Virtual Levels improves the running time just for the $PLDS$ problem.

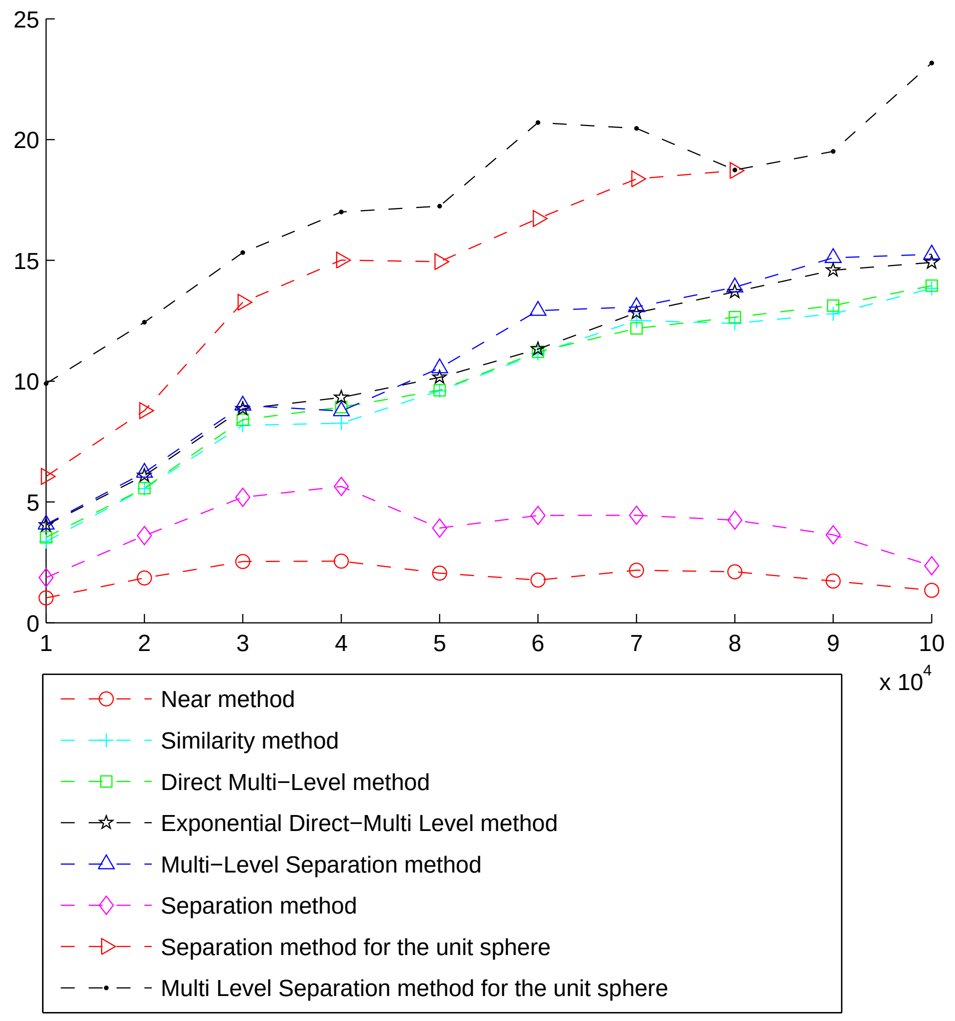

Figure 8: The running time for several methods compared with the naive search. The x axis represents the number of points, and the y axis represents the improvement compared with the naive search. The points have 128 dimensions, uniformly distributed on the unit sphere and have thresholds that are normally distributed according to N(0.5,0.1) such that the maximum threshold is bounded by the value 0.9.

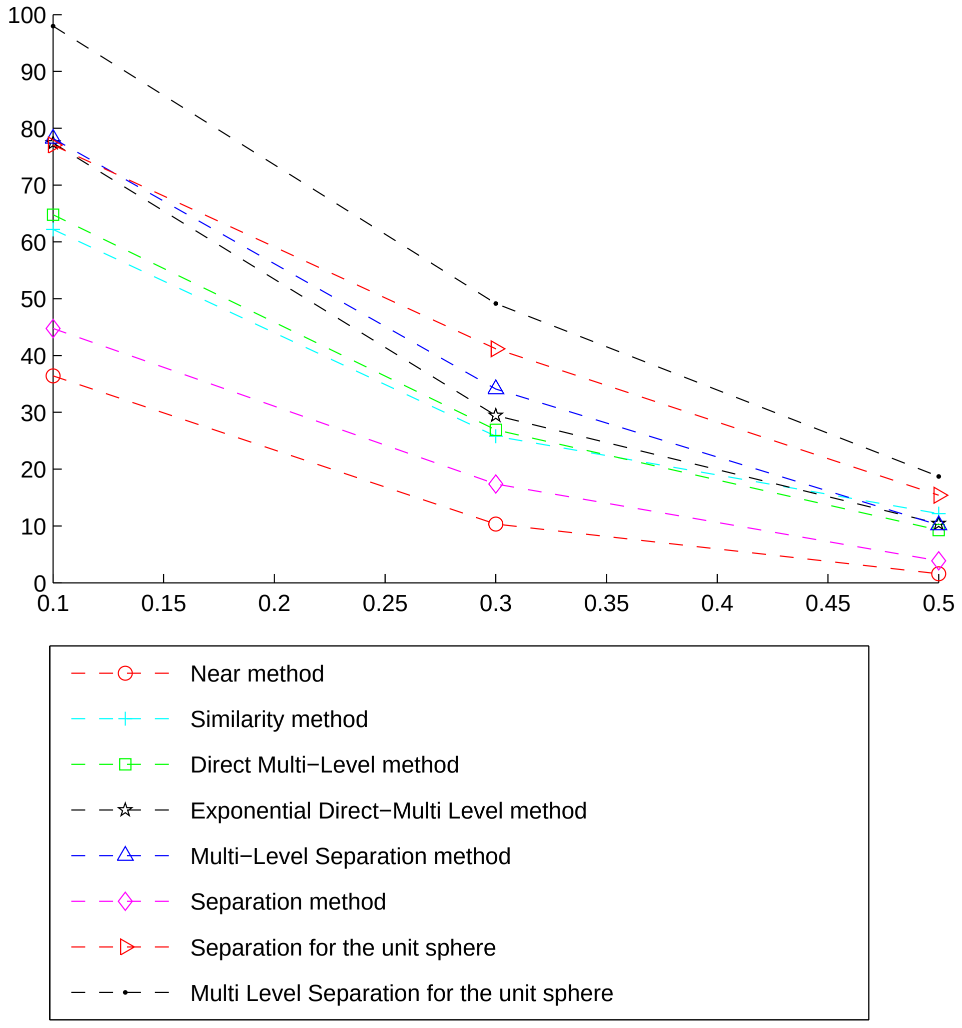

Figure 9: The running time for several methods compared with the naive search. The number of points is 50, 000 which are uniformly distributed on the unit sphere with thresholds that are normally distributed. The x axis represents the varying of the average thresholds. The y axis represents the ratio of the running time compared with the naive search.

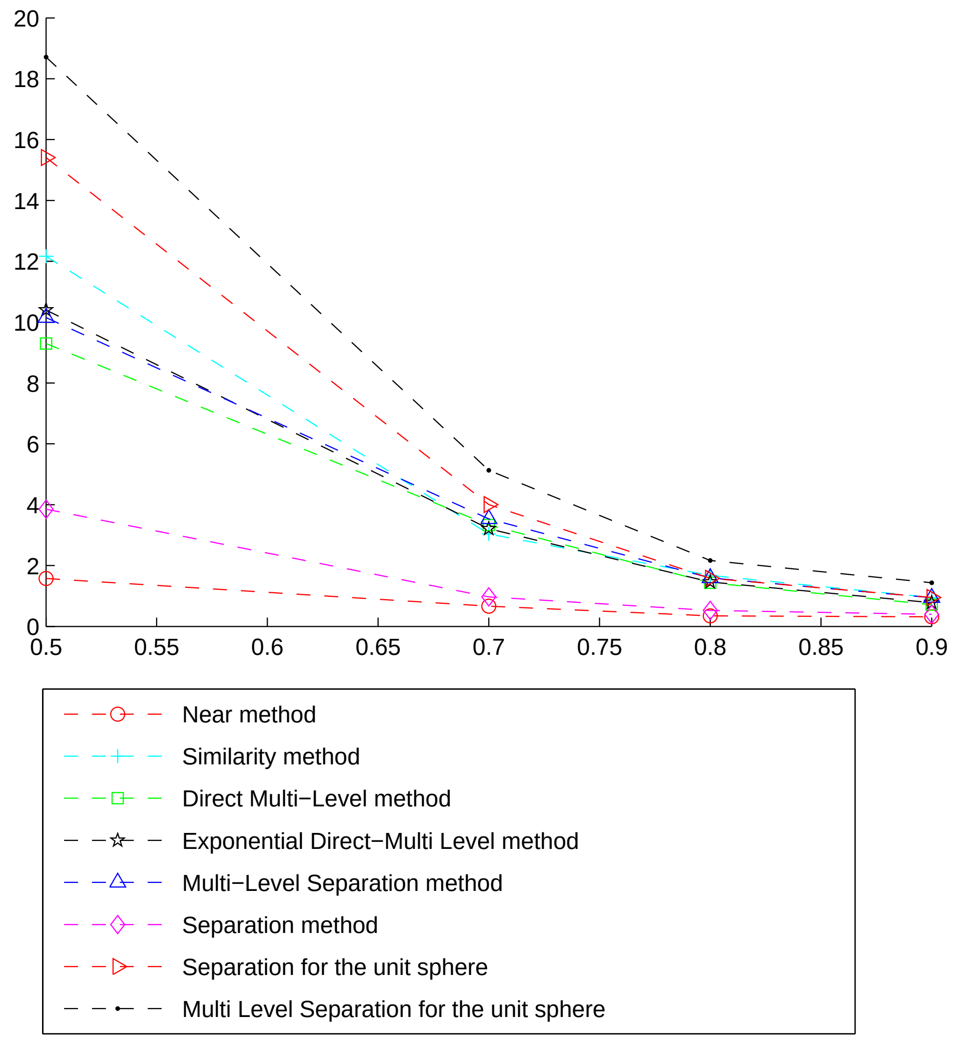

Figure 10: The running time for several methods compared with the naive search. The number of points is $50,000$ which are uniformly distributed on the unit sphere with thresholds that are normally distributed. The x axis represents the varying of the average thresholds. The y axis represents the ratio of the running time compared with the naive search.

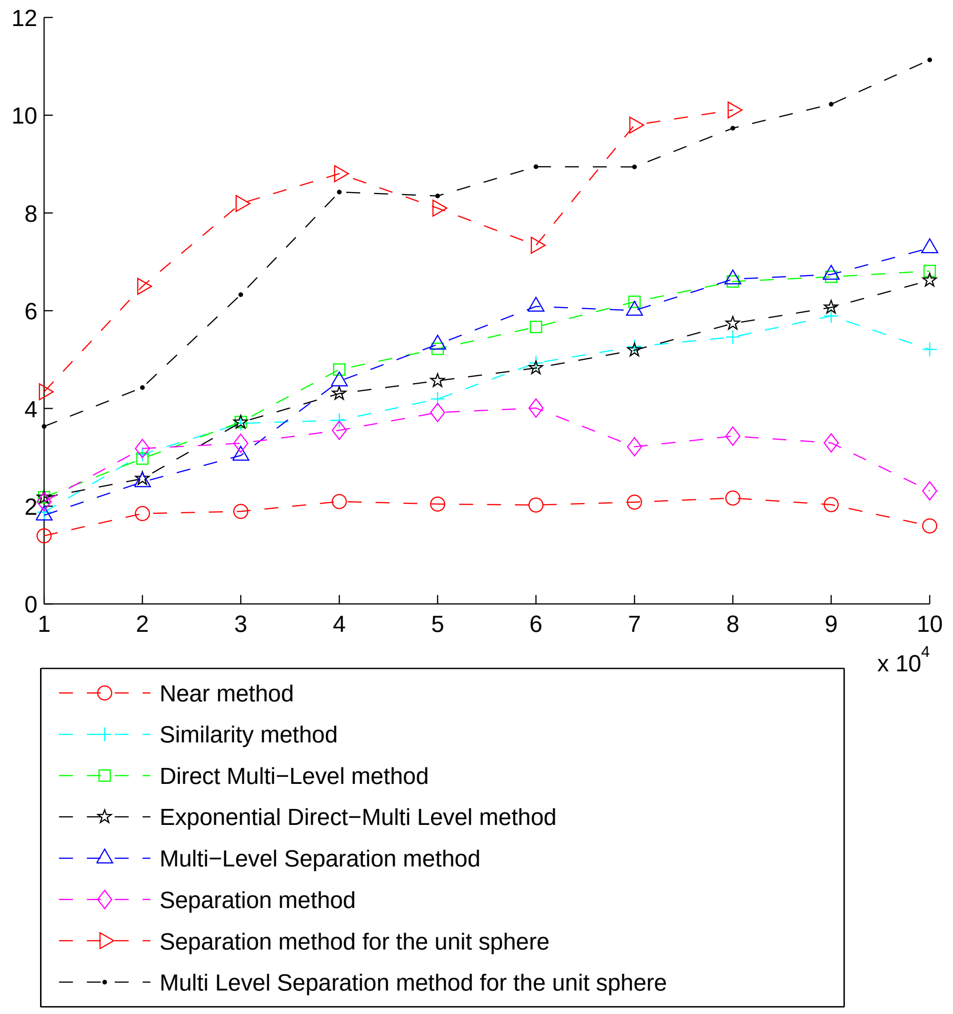

Figure 11: The running time for several methods compared with the naive search. The x axis represents the number of points, and the y axis represents the improvement compared with the naive search. The points have 128 dimensions, uniformly distributed on the unit sphere and have thresholds that are uniformly distributed in the interval [0.1,0.9]).

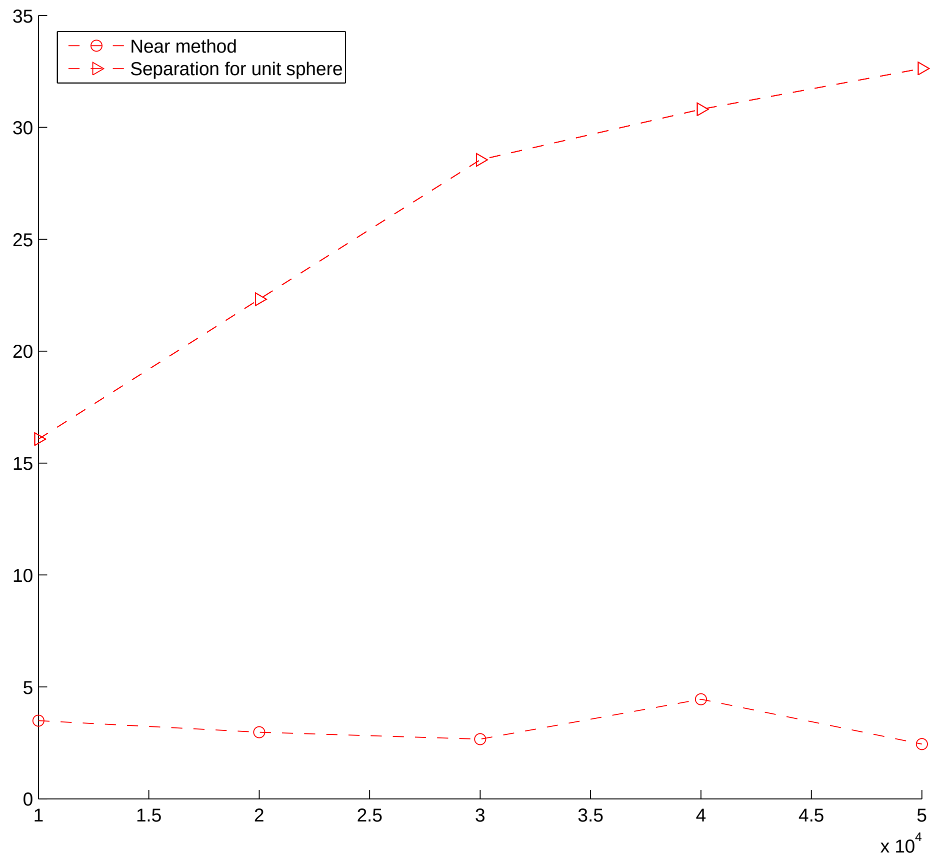

Figure 12: The running time for the Separation on unit sphere and the Near method compared with the naive search. The x axis represents the number of points, and the y axis represents the improvement compared with the naive search. The points have 500 dimensions, uniformly distributed on the unit sphere and have thresholds that are normally distributed according to N(0.5,0.1) such that the maximum threshold is bounded by the value 0.9.

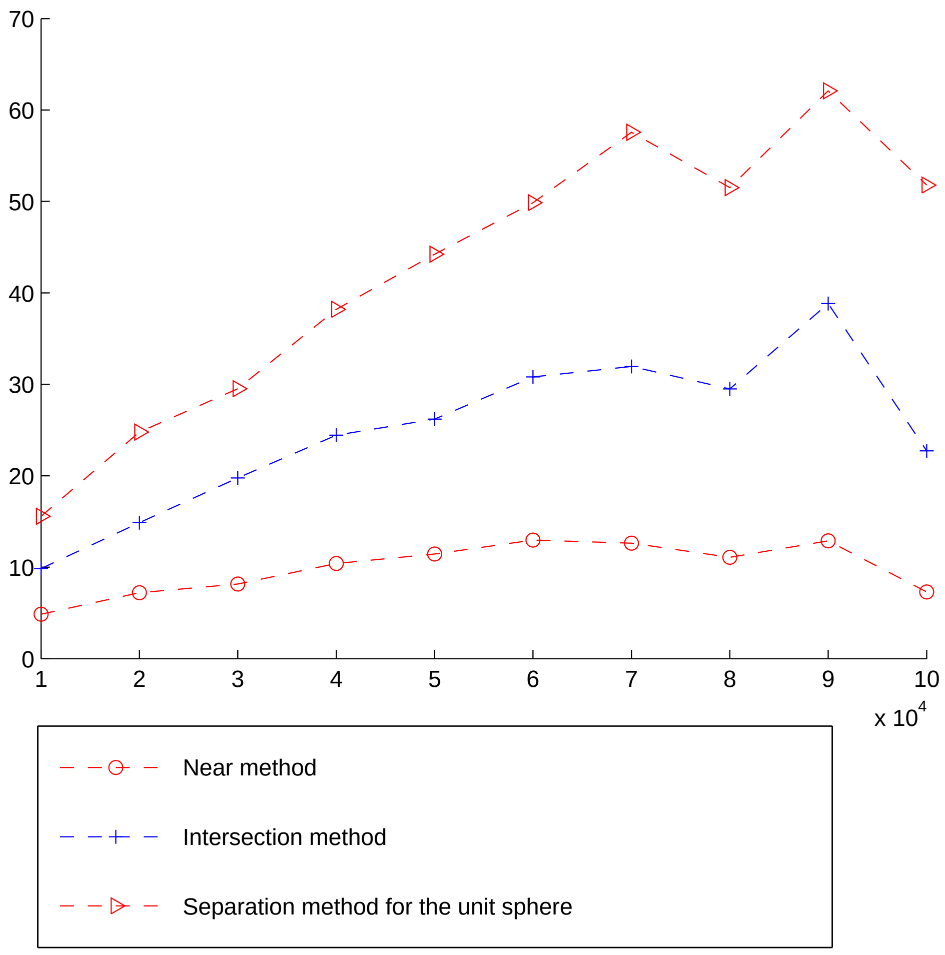

Figure 13: The running time for the Intersection method compared with the Near method. The x axis represents the number of data points, and the y axis represents the ratio running time compared with the naive search. The points are included in clusters such that each cluster contains 100 points; we varied the number of points on the x axis by adding more clusters. The thresholds distributed according to $N(0.1, 0.1)$.

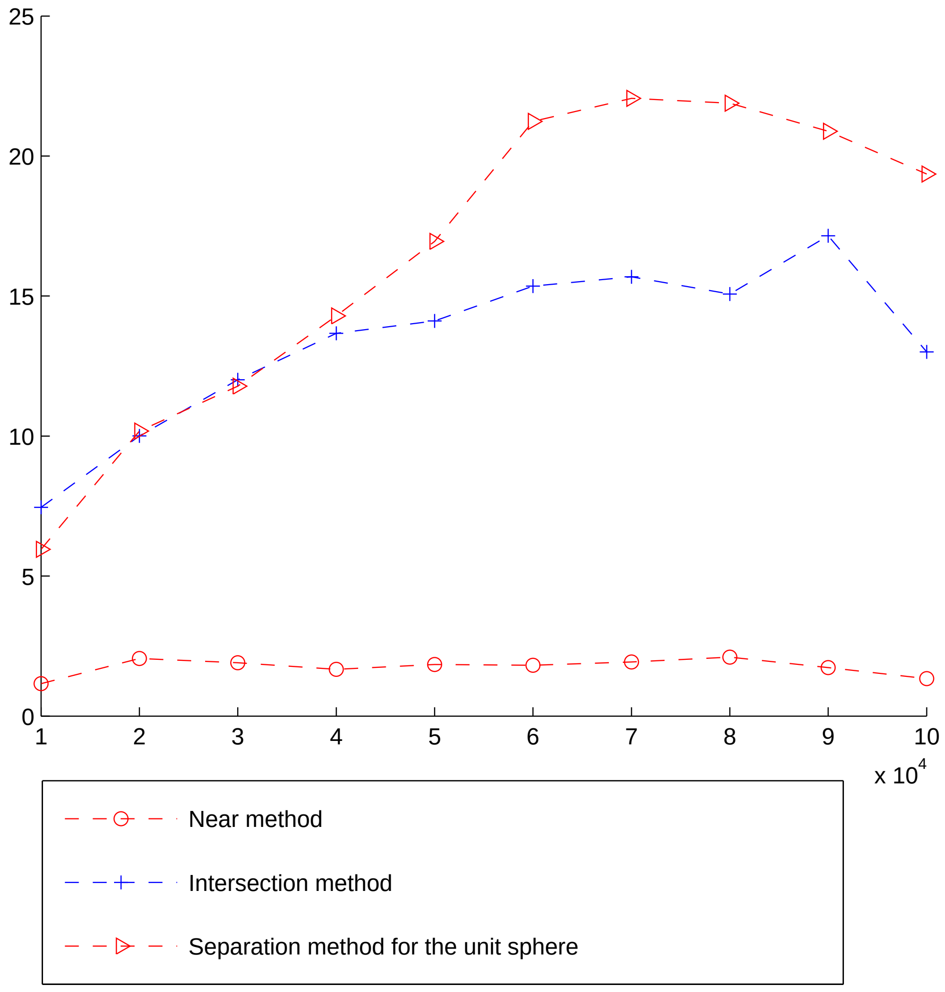

Figure 14: The running time for the Intersection method compared with the Near method. The x axis represents the number of data points, and the y axis represents the ratio running time compared with the naive search. The points are included in clusters such that each cluster contains 100 points; we varied the number of points on the x axis by adding more clusters. The threshold distributed according to $N(0.5, 0.1)$

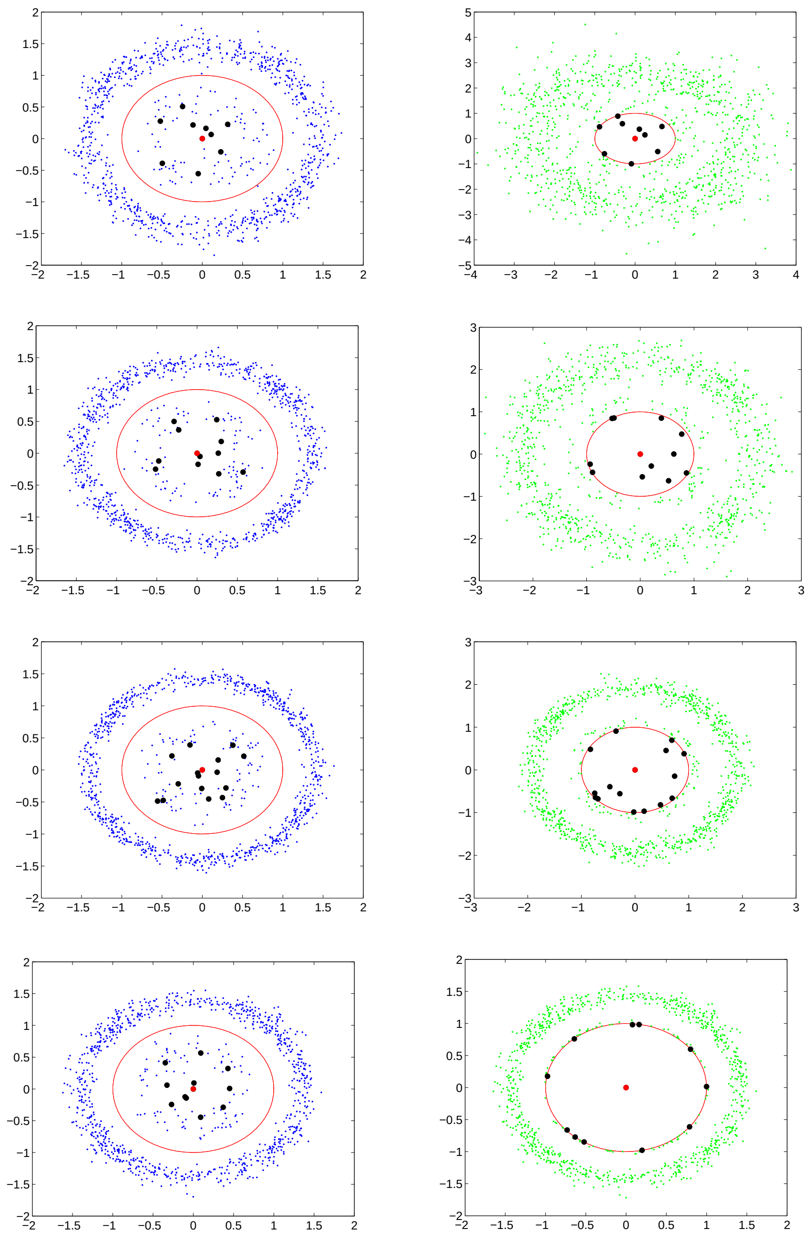

Figure 15: The Separation method for $L_{1/10}$, $l_{1/2}$, l_1, and l_5, respectively. The left plot shows the simulation of the distances from a query point of 895 points with 128 dimensions, randomly distributed on the unit sphere on a suitable l_p norm and 105 points distributed around the query point q inside the ball $B(q, r_{max})$. The right plot shows the simulation of the distances of the same points after applying the Separation method.