DATA TRANSFORMATION FOR IMPROVED QUERY PERFORMANCE

By

Alok Watve

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Computer Science

2012

ABSTRACT

DATA TRANSFORMATION FOR IMPROVED QUERY PERFORMANCE

By

Alok Watve

A database management system stores data in order to facilitate fast and efficient retrieval while executing queries. The typical queries that are run in a database can be classified into three broad categories. They are range queries, k-nearest neighbor (k-NN) queries and box queries. Implementation of a box query typically involves simple comparison of values in each dimension of the query feature vector with the corresponding dimension of an object in the database. Implementing range queries and k-NN queries, however, may be more involved due to computation of distances among the points. In this dissertation, we study mapping of one type of the query on to the other. From performance perspective, an index structure may favor one type of query over other. Hence, such a mapping provides a way of improving query performance and exploiting available system capabilities. It also highlights the relationships among the various types of queries.

Our first transformation maps a range query in L_1 space on to a box query. Since index pages of R-tree based indexing schemes are geometrically rectangles, this mapping provides a similar interface between the query space and the data space of each of the index page. In 2-dimensional space, the mapping is exact. However, it cannot be used directly for higher dimensional spaces. We propose a novel approach called disjoint planar rotation in order to extend the transformation in higher dimensions. We also develop a new type of box query, called pruning box query, which is equivalent to the range query in the original space. Our theoretical analysis shows that this mapping can improve I/O performance of the queries. Further, performance improvement increases with increasing number of dimensions. Due to the underlying similarity between range queries and k-NN queries, the proposed transformation can also be used to improve performance of k-NN queries. We also present a transformation to map box queries on to range queries in L_1 space. The inherent property of box queries to allow varying degree of selectivity along each dimension, poses some challenges for the transformation. We propose square tiling approach to map each box query on to a number of square box queries. Each of the square box queries can then be transformed into a range query. We demonstrate the effectiveness of this mapping using M-Tree.

Euclidean distance (or L_2 norm) is another popular distance measure. While exact mapping of range queries in Euclidean space to box queries may be challenging, using vantage point based indexing, it is possible to transform range queries into bounding box queries. Since, execution of bounding box queries is computationally more efficient than that of range queries, the transformation from range queries to bounding box queries can be used to improve query performance (i.e. the number of distance computations) in main memory databases. Existing work on vantage point based indexing uses data points from the database as vantage points. As a result the database becomes static and cannot allow dynamic insertions, deletions and updates. Further, Computational complexity of these vantage point selection schemes depends on the size of the database which can be a problem for large databases. We analyze the impact of vantage points on false positives and the number of duplicates; and present a heuristic algorithm for selecting vantage points in closed data spaces. As the vantage point selection is independent of data, the database allows dynamic insertions and deletions. Comprehensive experimental evaluation with several synthetic and real databases shows effectiveness of the proposed vantage point selection scheme.

To my parents Kishor and Jyoti Watve.
My wife, Shipra and my brother, Abhijit.

ACKNOWLEDGMENT

First and foremost, I would like to acknowledge my dissertation adviser Dr. Sakti Pramanik, who has provided constant guidance and encouragement throughout my graduate career.

I would like to express my earnest gratitude to my dissertation committee members Dr. George Stockman, Dr. Charles B. Owen, Dr. Qiang Zhu and Dr. James Cole for being there for me whenever I needed. Their expertise and suggestions have significantly improved this dissertation.

I also acknowledge our collaborators Dr. Shamik Sural, Dr. Sungwon Jung, Dr. Chad Meiners and Dr. Alex Liu for there support and guidance. I wish to acknowledge the support of the Michigan State University High Performance Computing Center and the Institute for Cyber Enabled Research. I also thank my lab colleagues Jignesh Patel, Srikanth Vudayagiri who made all these years thoroughly enjoyable.

Last but definitely not the least, I would like to thank of my wife Shipra and my parents Kishor and Jyoti, who always believed in me and loved me unconditionally.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

High dimensional feature vector databases are increasingly becoming popular in several applications such as content based image search, Geographical Information Systems (GIS), multimedia databases, etc. The most common queries executed in these databases can be roughly classified into three groups : Range queries (also known as similarity queries), k-NN (k-nearest neighbor) queries and Box queries. A range query returns all the data objects within a certain distance from the query object. A k-NN query returns k nearest objects from the query object while a box query returns all the objects whose individual dimensions satisfy certain criteria. We will formally define these three types of queries later in the chapter. Efficient implementation of these queries has been an active area of research for several decades. In disk-based database systems, the primary objective of a query implementation is to reduce the number of disk page accesses whereas in main memory database systems, the focus is to reduce the number of costly distance computations. This dissertation primarily focuses on data transformations that map range queries and k-NN queries onto box queries. As will be shown later, box queries have several implementational advantages which make them particularly preferable when database indexes are used. These advantages of box queries can be exploited to improve the run-time performance of the other types of queries.

In this chapter, we first present the basic concepts and notations that will be used throughout the dissertation. We then formally define the three types of queries.

1.1 Basic concepts and notations

Let $\mathbf{D}$ be the set of N objects in the database. Each object itself is a feature vector containing n real-valued dimensions and can be represented as a point in n-dimensional space $\mathbb{R}^n$. We denote a database object a as $a = (a_1, a_2, \ldots, a_n)$ where, $a_i (1 \leq i \leq n)$ is the projection of point a along dimension i (also known as i^{th} dimension of a). Without loss of generality, we assume that the data space is bounded to a unit (hyper)cube. In other words, $a_i \in [0, 1], \;\; \forall a \in \mathbf{D}$.

$d : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^+$ denotes the distance function that measures dissimilarity between the two points in the database. We denote distance between two points a and b as $d(a, b)$. Minkowski distance is a special type of distance measure denoted as L_p and formally defined as,

$$L_p(a, b) = \left(\sum_{i=1}^{n} |a_i - b_i|^p \right)^{1/p} \tag{1.1}$$

Here, p is called the order of Minkowski distance. In the limiting case where $p \rightarrow \infty$,

$$L_\infty(a, b) = \max_{i=1}^{n} |a_i - b_i| \tag{1.2}$$

Minkowski distances of order 1 and 2 (i.e. L_1 and L_2) are some of the most commonly used distance measures. Several authors [3, 11] show that in higher dimensional space,

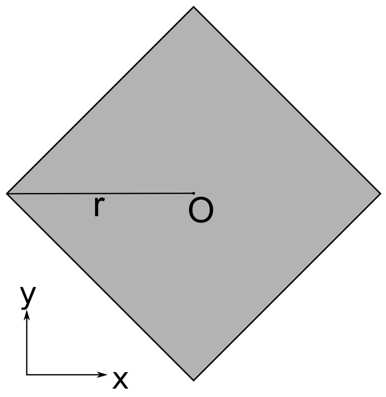
L_p norms with larger (i.e $p > 2$) values of p are not very meaningful. They propose a metric called "relative contrast" (which is essentially the difference between the distance of the farthest point and that of the nearest point with respect to a given point) to measure effectiveness of L_p norm. It has been observed that for higher dimensions and larger values of p, the relative contrast approaches zero highlighting the futility of the distance measures. For this reason, we primarily focus on L_1 (also known as Manhattan distance) and L_2 (also known as Euclidean distance) distance measures in this dissertation.

1.2 Metric space

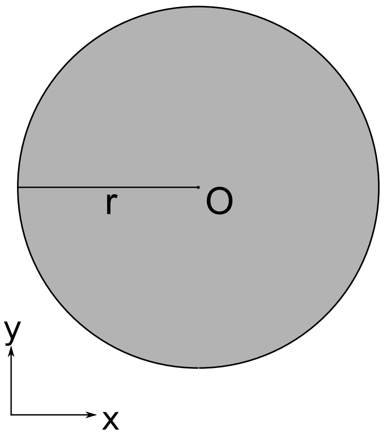
Space $\mathbf{D}$ is said to be a metric space when distance $d(a,b)$ between two points $a, b \in \mathbf{D}$, satisfies the following properties (called metric properties):

Symmetry :	$d(a,b) = d(b,a)$	$\forall \ a, \ b \in \mathbf{D}$
Reflexivity :	$d(a,b) = 0 \Leftrightarrow a = b$	$\forall \ a, \ b \in \mathbf{D}$
Non-negativity :	$d(a,b) \geq 0$	$\forall \ a, \ b \in \mathbf{D}$
Triangular Inequality:	$d(a,b) \leq d(a,c) + d(c,b)$	$\forall \ a, \ b, \ c \in \mathbf{D}$

Properties of metric distances can be used to prune the search space while searching in metric spaces. As an example, let us assume that we know the values of $d(a,b)$ and $d(b,c)$. Then the lower and upper bounds for distance $d(a,c)$ can be derived as $|d(a,b) - d(b,c)|$ and $(d(a,b) + d(b,c))$ respectively, without actually having to calculate the distance. These bounds are very useful in efficient implementation of queries which makes metric spaces (and metric distances) attractive in practice. In fact, many of the commonly used distance measures including the Minkowski norms indeed have metric properties.

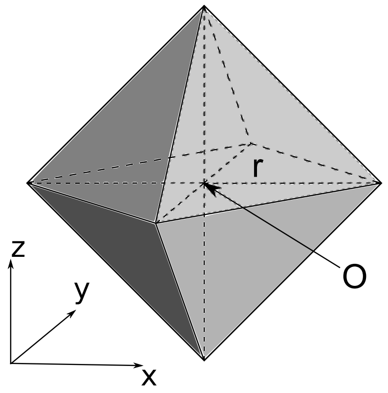


(a) Range query in L_1

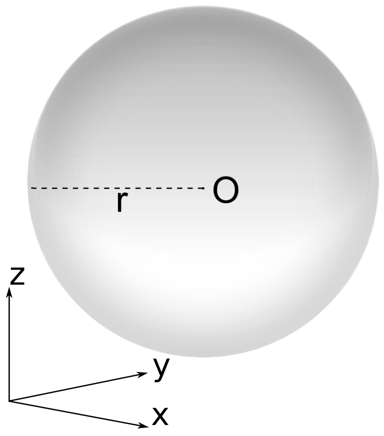


(b) Range query in L_2

Figure 1.1: Examples of range queries in 2 dimensional space



(a) Range query in L_1



(b) Range query in L_2

Figure 1.2: Examples of range queries in 3 dimensional space

1.3 Range query

We now formally define the range query. For the given database $\mathbf{D}$, the range query with radius r at point $a = (a_1, a_2, \dots a_n)$, denoted by $r@a$, is defined as,

$$r@a = \{b \mid b \in \mathbf{D} \ \wedge \ d(a, b) \leq r\} \tag{1.3}$$

Thus, a range query returns all the objects whose distance is less than or equal to the given radius r. Figure 1.1 shows example of range queries of radius r centered at O in 2 dimensional spaces using L_1 and L_2 norms. Notice that geometrical shape of an L_1 range query is a diamond while that of an L_2 range query is a circle. Figure 1.2 shows the geometrical shapes of corresponding range queries in 3 dimensional space.

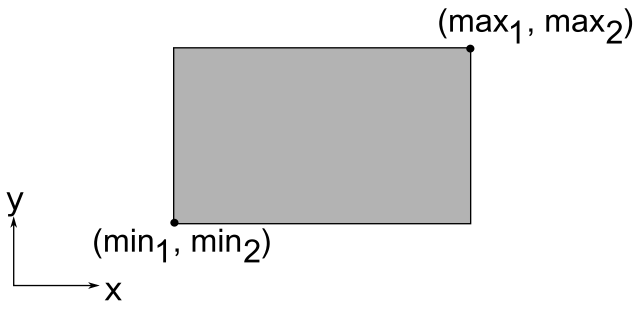
1.4 k-NN query

In a lot of applications, it suffices to retrieve only a few most relevant results (for example, when searching images or in standard web document search). As the name suggests k-nearest neighbor queries return the k objects that are the closest to the query object with respect to the given distance measure. Formally, a k-NN query at point $a = (a_1, a_2, \ldots, a_n)$ denoted by $k@a$ is defined as,

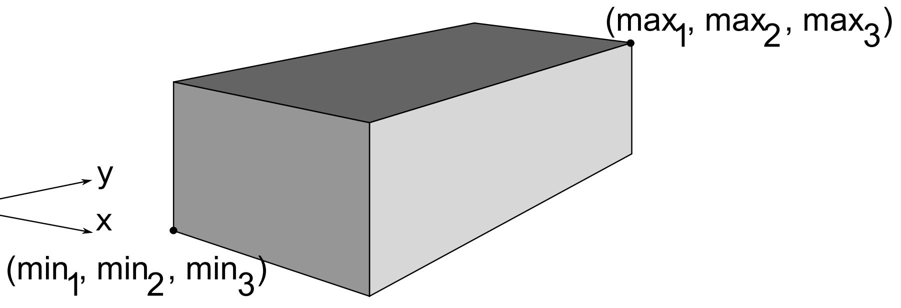
$$k@a = \mathbf{D_k} \left| \begin{array}{l} \mathbf{D_k} \subseteq \mathbf{D} \;\wedge \\ |\mathbf{D_k}| = k \;\wedge \\ \forall\, b \in \mathbf{D_k}, \forall\, c \in \mathbf{D} - \mathbf{D_k}, \quad d(a,b) \leq d(a,c) \end{array} \right. \tag{1.4}$$

Here, $|\mathbf{D_k}|$ denotes the cardinality of the set $\mathbf{D_k}$.

Although we identify k-NN queries as a separate class of queries, from implementation point of view, they can be viewed as a variant of range queries. We will revisit implementation of k-NN query as a range query later in section 3.8.



(a) Box query in 2-D



(b) Box query in 3-D

Figure 1.3: Examples of box queries in 2 and 3 dimensional spaces

1.5 Box query

Box query is a query that returns all the objects that satisfy individual criterion (i.e. range of permitted values) for each dimension. Let $\min_i$ and $\max_i$ be the minimum and maximum value allowed on dimension i $(1 \leq i \leq n)$ and let $\nu_i = [\min_i, \max_i]$. Then the corresponding box query, represented by $b@(\nu_1, \nu_2, \dots \nu_n)$, is defined as follows:

$$b@(\nu_1, \nu_2, \dots, \nu_n) = \{a \mid a \in \mathbf{D} \quad \wedge \quad a_i \in \nu_i \text{ for } 1 \leq i \leq n\} \tag{1.5}$$

Thus, a box query returns all the data point whose dimensions lie in the respective interval of values.

Figure 1.3 shows examples of box queries in 2 and 3 dimensional spaces respectively. An important property of box query is that shape (or the query space) of the box query only dependent on values allowed in each dimensions and is independent of the distance measure used for the space (i.e. box query in L_1 is same as the box query in L_2).

1.6 Motivation for query transformations

For efficient execution of queries in a very large database, usually a multi-dimensional index is created for the database and queries are implemented using this index. Note that execution efficiency may refer to the number of disk accesses (in disk based indexing) or the amount of CPU computations (in main memory based indexing). In a typical tree based index, a query execution algorithm attempts to find if the data space of a node in the index tree overlaps with the query space. If there is an overlap then that node is searched further; else that node can be pruned from the search. Effectiveness of an index depends on its ability to prune unwanted search branches which in turn depends on the geometry of the query space and data spaces of tree nodes. This results in an index tree favoring one type of query over another. In the proposed work we study relationships among the different types of queries with special focus on mapping one type of query on to another. Such mapping is important for the following reasons :

1. It enables a more efficient implementation of the queries by mapping them to the type of query favored by the index. In case of disk based database systems, this translates to lesser number of disk page accesses, and for main memory database systems, it results in a lesser number of distance computations.

2. It provides a tool to implement both types of queries using the same query execution engine. This may eventually obviate the need to explicitly support both the types of

queries in a database management system.

3. It lets the user take advantage of an indexing scheme (or any other optimization) designed exclusively for a particular type of query.

4. It lets us gain insights into the relationships among various types of queries and the query types and the index. This may lead us to better indexing schemes.

1.7 Outline of the dissertation

The rest of the dissertation is organized as follows: In chapter 2, we present the prior work related to this research and discuss some of the relevant indexing schemes. Chapter 3 presents technique for mapping L_1 range queries onto box queries. Chapter 4 discusses inverse mapping of box queries on to L_1 range queries. Chapter 5 presents some of the possible transformation approaches for L_2 queries in disk-based databases. Chapter 6 focuses on the special type of transformation based on vantage points, in context of main memory databases. Conclusion and future work follow in the last chapter.

Chapter 2

Related Work

There has been a lot of work on multi-dimensional indexing for database systems and executing queries using such indexes. However, use of data transformation to improve query performance is a relatively new research topic. In this chapter we discuss the existing work on transformation based data access methods. We also present an overview of some of the popular indexing schemes that are of particular interest in the context of this dissertation. We begin with various transformation approaches proposed for efficient queries and data retrieval.

2.1 Data transformation techniques

A lot of the existing work on data transformation for efficient retrieval primarily focuses on reducing dimensionality of the data. It is a known fact that many of the existing search techniques which work well in the lower dimensional space, break down when dimensionality of the data increases. In fact, Weber et al. [65] show that some of the traditional indexing schemes are outperformed by linear scan (i.e. storing data in flat files without any indexing) when the number of dimensions is greater than 10. This fact, known as dimensionality curse [8] is the primary motivation for reducing the data dimensionality. Many of the real world data have dimensions which are not independent of each other. Hence, the extrinsic (or the apparent) dimensionality of the data is much more than their intrinsic dimensionality. Avoiding this redundancy is also another reason why one may attempt dimensionality reduction. Data dimensionality is also an important parameter in any multi-dimensional indexing scheme as it directly affects fan-out of a typical disk based index. Hence, a lot of work has been done to reduce dimensionality of the data.

The linear algebra techniques of Principal Component Analysis (PCA - also known as Karhunen-Loeve theorem) [40] and Singular Value Decomposition (SVD) [27] are commonly used for dimensionality reduction. In general the source dimensions of the data may not be orthogonal. PCA effectively transforms the data into another space such that all the dimensions are orthogonal. Further the dimensions can be ordered in decreasing order of data variance (i.e. data has the maximum variance along the first dimension and so on). A user can then select the desired number of dimensions such that most of the variance is captured while considerably reducing the dimensionality of the data. SVD decomposes a feature matrix M into components $M = U\Sigma V^T$ such that U, V are orthonormal and Σ is diagonal matrix of *singular values*. Some of the smallest singular values can then be dropped to reduce the dimensionality of the data without hurting the overall data variance.

Some of the earliest applications of dimensionality reduction for efficient database indexing and retrieval are the text retrieval systems proposed in [24, 36]. The authors use an SVD based scheme called latent semantic analysis [20] to reduce the dimensionality. The QBIC (Query By Image Content) system [52] developed by IBM uses SVD for extracting relevant dimensions from efficient retrieval of image data. Note that in dynamic databases (where the data may change anytime due to insertion, deletions or updates), PCA or SVD analysis needs to be redone with changing databases for accurate results which can be very expensive. A technique for incremental SVD (i.e. using past properties of data to simplify calculation of SVD of new data) is proposed by Ravi Kanth et al. [56]. Aggarwal [2] presents an analysis of existing dimensionality reduction on quality of the results. More examples of dimensionality reduction techniques for data retrieval can be found in [17, 64].

TV-tree [38] proposed by Lin et al. is an indexing schemes which tries to minimize the number of features used to create tree. Essentially a feature is used for indexing only if all the features used so far cannot distinguish between the objects. Conceptually this is similar to starting with projection of data in low dimensional space and then gradually adding dimensions as they are required. For each child disk page, a bounding sphere is maintained by the parent such that all the objects in the child page are contained in the bounding sphere. Some other methods for dimensionality reduction can also be found in [21, 43, 51].

As an extreme case of dimensionality reduction, a lot of work has been done in transforming the multidimensional data to 1-dimensional data. These methods use space filling curves (such as Z-curve [50] or Hilbert curve [32]) which allow them to order high dimensional feature vectors. Such methods employing Z-curve for executing box queries are proposed in [54, 58, 6]. An interesting clustering based scheme called iDistance [39] is proposed by Jagadish et al. They first cluster all the data points. Then each point is given a simple numeric identifier which is derived from its cluster and its similarity with respect to the cluster center. As each multi-dimensional point is mapped to a one dimensional numeric value, it can be indexed using one dimensional indexing scheme such as B^+ tree.

In a rather counter intuitive way, some indexing schemes have been proposed which map data from lower dimensional space to higher dimensional space. Some of these methods [22, 33, 44] transform objects (or polygons) into higher dimensional points and then use one of the established point access methods such as the the grid file[53]. A detailed discussion on all the transformation based data access schemes can be found in a survey by Gaede et al.[25].

Linear transformation is a well known technique in linear algebra [45]. However, its applications to database queries have not been explored much. To the best of our knowledge, other than our work in [55], there is no other work in disk-based indexing on mapping of range queries to box queries using data transformation. However, there has been some work on approximating range queries using minimum bounding box queries. Although this mapping has a lot of false positives, it guarantees all the true positives. In the next section, we look at some of the existing work in this area.

2.2 Vantage point based transformation

In main memory database systems, data transformations based on vantage points (also known as pivots or split points or reference objects in the literature) are particularly popular. The main idea is to express each data point from the original space in terms of its distances from a set of vantage points. In metric spaces, properties of the distances can then be used to prune the search space. Using this transformation, range queries in the original space are mapped to bounding box queries in the transformed space. The basic idea behind this type of implementation of range query using box query is presented by Uhlmann[60, 61]. Most of the schemes in this domain use some kind of tree based indexing either implicitly or explicitly. The existing work in this area can be divided into two categories based on whether the index is built in the original space or in the transformed space. It should be noted that although transformation and mapping of range query to bounding box query is an important aspect of these schemes, the selection of the right vantage points is the primary research focus of this dissertation.

One of the earliest tree-based methods using the concept of vantage points is the VP-tree [66]. It uses only one vantage point per node of the tree. Vantage points are chosen in such a way that each node partitions the data space in almost balanced partitions. The author suggests that good vantage points are the points closer to the corner of the subspace and the vantage point selection heuristic (choosing the point with maximum spread i.e. second order moment about median distance) is designed to address this. GNAT [14] is another tree based indexing scheme based on the variable number of vantage points (split points) per node. Different set of vantage points are chosen at each level of the tree. The basic strategy for vantage point selection is to choose points which are relatively the farthest from a set of vantage points. Each vantage point defines a subtree of the index. Each point in the node is assigned to the subtree of the nearest vantage point. Maximum and minimum distance of each point in the subtree from the vantage point is recorded. In metric space, this information can be combined with triangle inequality to decide which subtrees need to be searched. The construction of GNAT is more expensive than VP-tree due to the amount of processing required to generate the vantage points. However, executing a range query in GNAT requires fewer distance computations than the VP-tree. MVP-tree [12, 13] is a generalization of VP-tree into a multiway tree. MVP-tree uses two vantage points at each node. Each vantage point can be used to partition the space in m partitions. The second vantage point partitions each partition of the first vantage point in m partitions. Thus the fan out of an internal node is m^2. The vantage points themselves are chosen such that the second vantage point is the farthest data point from the first. The first vantage point can be chosen at random or by choosing the farthest point from a random data point. MVP-tree computes a lot of distances during the tree construction time. Some of these distances are cached in the nodes which avoids some distance computations at the query time. As MVP-tree allows for large node fan-out, it can be implemented as a main memory tree or a disk based tree. Mao et al. [47] use a PCA based vantage point selection technique in MVP-tree frame work. They first select a set of candidate vantage points (outliers found using Fast Fourier Transform - FFT). PCA of this candidate set is then used to identify dimensions which have the maximum variance. Data points which have the maximum projection along these dimensions are then chosen as the vantage points. They also use PCA to identify intrinsic dimensionality (hence the best choice of the number of vantage points) of the space.

The methods discussed so far select vantage points at each node of the index tree. Thus, vantage point selection considers information local to the index node. The index itself is built in the original space. Hence, the partitions created by these methods are geometrically very complex having (hyper-)spherical boundaries. These shapes do not interface very well with spherical range query. However, by using global vantage points and by mapping range queries on to bounding box queries through space transformations, it is possible to have (hyper-)rectangular partitions interfacing with (hyper-)rectangular queries. This simplified interface may greatly benefit efficient execution of queries. Hence, lately the focus have shifted to applying space transformation using global vantage points.

One of the first approaches using global vantage points is proposed by Vleugel et al. They use balanced box tree [5] to implement their strategy, but in theory, it can be used with any multidimensional main memory indexing scheme. The authors use different vantage point choice strategy for different databases, which can become a problem for generalization of the scheme. For one of the datasets, they propose a Max-Min heuristic in which vantage points are chosen such that they are relatively the farthest from each other (similar to GNAT). A more generic vantage point selection algorithm for any type of data, is proposed by Hennig et al. [31]. For each set of vantage point, they define a loss function for query q and resultset R as D(q, R), which is the L2 distance between the vantage space representation of the two objects. Their algorithm attempts to minimize the overall loss by incremental selection of locally optimal vantage points. Around the same time, Bustos et al [16] observe that vantage point with larger pairwise mean distance tend to have lesser false positives. They present three strategies to exploit this property. These strategies are, (i) Generate several random samples and choose the best one (i.e. one which has maximum mean distance) (ii) Incrementally select vantage point to maximize mean distance at each step (iii) Select a random sample and replace the points which have the least contribution toward mean distance. Although authors have compared the three strategies amongst themselves in detail, they have not compared these methods against any of the established (or published) methods which makes it difficult to gauge the effectiveness of their proposal. Brisaboa et al [15] propose sparse spatial selection (SSS) technique for vantage points. They try to ensure that the vantage points are sufficiently far apart by requiring each vantage point to satisfy certain minimum distance threshold of $M\alpha$ where, M is maximum possible distance ($= \sqrt{n}$ for n dimensional unit cube in L_2) and α is a control parameter which is empirically determined to be between 0.35 to 0.4. The authors propose increase the vantage point set as long as no new vantage points can be added without hurting the distance criterion. Hence, the number of vantage points used is more than the dimensionality of the original space, which essentially means that dimensionality of the transformed space is greater than that of the original space.

Recently, Van Leuken et al. [62] have proposed two novel heuristics for choice of good vantage points. We will discuss these heuristic in more depth later in chapter 6. Here we just briefly describe their heuristics. The first heuristic suggests minimizing spacing variance. Spacing is defined as the difference between two successive distances in transformed space. The authors propose that minimizing spacing variance ensures that points are spread uniformly in the vantage space. Their second heuristic is to minimize correlation of distance vectors of two vantage points. A pair of vantage points with high correlation implies that that at least one of the points is redundant. They propose a random sampling based algorithm which uses these two heuristics. Experimental results show these heuristics to be better than the earlier methods by Henning et al [31], Bustos et al. [16], Brisaboa et al. [15]

A related space-transformation based approach for approximate range queries and $k-$NN queries in disk based index is proposed in [4] and [18]. Although the basic premise is same as the other vantage point based methods, the authors use ranks of the data points (in terms of distances from reference objects) as representatives of the records. Several heuristics have been used to improve recall and reduce disk accesses. This idea can be used with any of the distance measures such as Spearman Footrule distance, Kendall Tau distance and Spearman rho distance [18, 26]. However, these methods are sensitive to the statistical characteristics of the database and a lot of learning may be needed to determine values of some run-time parameters. Further, the number of vantage points required for reasonable recall increases with increasing database size and may not work well with dynamic databases. Another vantage point selection scheme for disk-based indexing of protein sequences is proposed by [63]. The authors choose the set of vantage points which has maximum variance. This set if further optimized by evaluating pruning power of each vantage point and if possible replacing it with a point with better pruning power. Note that use of this approach require some prior knowledge about query pattern in order to estimate the pruning power.

2.3 Indexing schemes for multidimensional data

In this section we review the existing multidimensional indexing schemes that will be used (and hence are relevant) in this dissertation. We also briefly describe implementation of different types of queries on these indexes. For more comprehensive discussion on index schemes for various types of queries, reader is directed to surveys by Samet et al.[35] and Ilyas et al.[37]. We begin with R*-tree [7] which is one of the most popular indexing scheme based on R-tree [30].

2.3.1 R*-tree

R*-tree[7] is a height balanced tree which can store both multidimensional points as well as bounding boxes. The data objects are stored in the leaf nodes. Each directory node contains bounding boxes (or MBRs - Minimum Bounding Rectangles) of child nodes along with pointers to the chide nodes. Geometrically every disk page in an R*-tree is a (hyper)rectangle. The fundamental property of R*-tree (and any R-tree based index) is that all the nodes in the subtree rooted at node N have their bounding boxes (in case of directory nodes) or data objects (in case of leaf nodes) completely inside the bounding box of N. In other words, if node N has interval $[\min_i(N), \max_i(N)]$ along dimension i then for each child node C with the interval $[\min_i(C), \max_i(C)]$ along dimension i, we have, $\min_i(N) \leq \min_i(C) \leq \max_i(C) \leq \max_i(N)$. The maximum fan out of the tree M is calculated so as to pack maximum possible number of entries in a single page of the index. One tuning parameter for ensuring good disk utilization in R*-tree is minimum fan out $m(\leq M/2)$. R*-tree insertion algorithm guarantees that each page in the tree contains at least m children (i.e. disk utilization of m/M). R*-tree is built bottom up meaning that any insertion or deletion is first performed in the leaf node and then the change is propagated all the way to the root.

Insertion algorithm in R*-tree uses three main heuristics namely, minimum overlap, minimum perimeter and maximum storage utilization. Whenever, there are multiple candidate nodes for insertion of a data point, it first chooses the one which results in minimum overlap among the MBRs of the nodes. This heuristics attempts to minimize the number of search paths required to be traversed for searching, thereby improving the search efficiency. In case of tie, it chooses the one that results in minimum perimeter. This heuristic attempts to create more squarish MBRs. If there still exist multiple candidate then it chooses the node that results in maximum storage utilization. Higher storage utilization decreases the total number of nodes in the tree there by reducing the disk space requirements. In addition to these heuristics, R*-tree periodically reinserts some of the data points in order to stabilize the tree against different initial configurations. Algorithms for implementing various types of queries are described below.

2.3.1.1 Box query implementation

R*-trees naturally support box queries as each disk page itself is represented as a box. At each node N, the query box compared against the bounding box each child of N to check if they overlap. Note that, the two boxes will overlap if and only if their intervals along each dimension overlap. Each child whose bounding box overlaps with the query box is recursively searched.

2.3.1.2 Range query implementation

Range query implementation in R*-tree is based on the fact that if the bounding box of a node overlap with the query sphere then that node may contain data objects satisfying the query. For node N, we can find minimum distance of the bounding box of each of its children from the query center. If this distance is less than query radius then that child is searched recursively, otherwise it is pruned.

2.3.1.3 *k*-NN query implementation

Roussopoulos et al. [57] proposed some of the earliest heuristics for executing k-NN queries in R*-tree. The basic idea is to keep track of the distance d_k of the current k^{th} neighbor from the query point. If the minimum distance of the bounding box of a node N is less than d_k then that node may potentially contain a point nearer than the current k^{th} neighbor. Hence, that node needs to be searched. Hjaltason et al. [34] showed that for k-NN queries, the order in which nodes are searched is critical. Instead of traditional depth-first search, they propose using best first search approach which seems to be more effective in pruning the search faster. The basic idea in best-first search is to maintain a pool of all the candidate nodes, and pick the one that is currently the closest to the query point (irrespective of its position in the tree). As the better k-NN candidates are found, the pool of nodes is refined.

2.3.2 Packed R-tree

Packed R-tree [41] is a variation of R-tree that is suitable for static databases. Static databases are the databases which do not have any updates or deletions and insertions are done at the beginning when the index is built. Thus, all the data is available prior to building the index. Dynamic index schemes such as R-tree or R*-tree do not have any control over order or distribution of data. Packed R-tree however can organize the data such that it suits the index better. It pre-sorts the data using space filling curves such as Z-order [50] or Hilbert curve [32]. The sorting phase ensures spatial locality based ordering of the data objects (i.e. data that are close to each other in the order are spatially close). The ordered database objects are then maximally packed into the leaf nodes and leaf nodes are then maximally packed into the higher level directory nodes. The last step is repeated until a single root node is obtained. Following the R-tree scheme, each node is represented in terms of its MBR in its parent node, hence, geometrically each node is a (hyper)rectangle. The primary advantage of this scheme is that disk utilization in almost 1 (i.e. each node is filled to the maximum capacity). As Packed R-tree is a variant of R-tree, the algorithms to run box queries, range queries and k-NN queries are similar to those mentioned in the previous section.

2.3.3 M-tree

The indexing schemes discussed so far are mainly based on partitioning of the space based on spatial locations of objects. However, there has been a lot of work in design and implementation of indexing schemes that are based on distances among the objects. Vantage point based indexes described earlier in this chapter fall in this category. For disk based databases however, M-tree [19] is one of the most popular distance based indexing scheme. M-tree is a height balanced tree built in bottom-up fashion. Each node in an M-tree contains a point (center) and a distance (r) for each of its children. All the data points in a child node are guaranteed to be within distance r from the center of that node. Geometrically, each node in an M-tree is a (hyper)sphere. Insertion in an M-tree proceeds as follows: For each point to be inserted, first we find if there is already a node such that, the point lies within the radius r of the node. If no such node can be found, then the node which requires minimum increase in its radius is considered. If the insertion causes the node to overflow, it is split to create two smaller nodes such that the two new nodes follow preset disk utilization constraint. Changes in the lower level nodes due to insertion percolate to the higher level nodes. The authors propose a number of heuristics for splitting the overflowing node based on the distance between the new centers, radii of the new nodes and distribution of points in the new nodes. According to the authors, the heuristic named mM_RAD that minimizes the maximum of the radii of the new nodes provides the best performance and is applied in the implementation used in our experiments.

2.3.3.1 Box query implementation

The original M-tree is designed for range queries and k-nearest neighbor queries. However, it is possible to implement box queries in it. The main idea is to figure out if a node can overlap the query box or not. For each child of node N, we first find distance of the nearest point of the query box from the center of the child node. If this distance is less than the radius of the child then that node is searched recursively else it is pruned. We would like reiterate that although M-tree can be used for box queries it is not the best index for this type of queries.

2.3.3.2 Range query implementation

M-tree supports range queries naturally. Recollect that M-tree is designed to be used for metric spaces, hence, the distance measure used for M-tree satisfies the metric properties. Each node in an M-tree contains centers and radii of all its children. For executing a range query, we first find the distance between the query center and the center of a child node. By triangular property if this distance is greater than sum of the two radii (query radius + child node radius), then the node can be safely pruned because it cannot contain any data objects that satisfies the range criterion. Any node whose distance is less than the sum of the two radii is searched recursively. The original M-tree algorithm also keeps track of distance of a node from the parent. This distance can be used to avoid unnecessary distance computations thus reducing the CPU cost of the implementation.

2.3.3.3 k-NN query implementation

Implementation of k-NN query in M-tree is conceptually similar to that in R*-tree. Initially, we do not have any estimate on the distance of the nearest neighbors hence we proceed using the best first search algorithm to get the first set of K-nearest neighbors. The distance of the k^{th} neighbor, d_k provides the distance threshold that each subsequent data page must satisfy in order to be considered as a search candidate. Again, using the triangular inequality property of the metric space, a node can be safely pruned if distance between the center of the node from the query point is greater than the sum of the radius of the node and the distance of current k^{th} neighbor. If a point whose distance from the query point is less than d_k is found, then it replaces the current farthest point in the k nearest neighbors and the threshold distance d_k shrinks.

2.3.4 KD-tree

KD-tree[9] is one of the earliest multidimensional indexing scheme for main memory based databases. It is a space-partitioning binary tree that recursively divides n dimensional space by $n-1$ dimensional hyperplanes. Function of each node in the KD-tree is two fold. First, it stores one data point, second, it provides a splitting point for next level. The node splits the space along a certain dimension, called splitting dimension, which is chosen in round robin fashion. The root splits the space along first dimension. All the data points whose first dimension is less than that of root node are indexed in the left subtree of the root. All the points whose first dimension is greater than or equal to the root node are indexed in the right subtree of the root. Although not guaranteed to be balanced, it can provide average run time of $O(\log n)$ for insertion and searching.

2.3.4.1 Box query implementation

Box query (referred to as region query in the original paper) implementation in a KD-tree is straightforward. At each node, the algorithm has to find answers to two sub-problems. first, it has to check if the data point corresponding to the node itself is contained in the box query. If it is, then it is added to the resultset. Second, the algorithm has to determine if the query box overlaps with left or right subtree (which in fact is a hyper rectangle). A subtree is searched only if the hyper rectangle corresponding to the subtree overlaps with the query box.

2.3.4.2 Range query implementation

Range query implementation in a KD-tree is conceptually similar to box query in the sense that at each node range query algorithm has to solve the same sub-problems. However, the algorithm may have to keep track of a lot more information if it has to prune the search effectively. Note that, by default KD-tree does not keep explicit information about the subspace corresponding to each node. Algorithm for range query in KD-tree needs to keep track of this information. At each node, it first determines if the data point corresponding to the node should be included in the resultset. Next, it checks if the query sphere overlaps with the sub-spaces of left and right subtrees to decide if they need to be searched. Each overlapping branch is then recursively searched.

2.3.4.3 *k*-NN query implementation

An algorithm to implement *k*-NN queries using KD-tree is presented in [23]. First, the algorithm searches for the query point as if it were to be inserted. The leaf node corresponding to the insertion gives is used as the first estimate for the distance of the nearest point. O The basic idea is to estimate if a particular sub-tree can have a neighbor or not. This is done by constructing a sphere whose radius is equal to the distance of k^{th} neighbor. If this sphere does not overlap with the sub-space of a branch, then that branch can be safely pruned. Else it needs to be searched.

Chapter 3

Transformation From L_1 Range Query To Box Query

In this chapter, we present transformation of L_1 range queries to box queries. The objective of transformation is to take advantage of the particular shape of the box query for better index performance. We first present the transformation in 2-dimensional (2-D) space and then extend it for higher dimensional spaces.

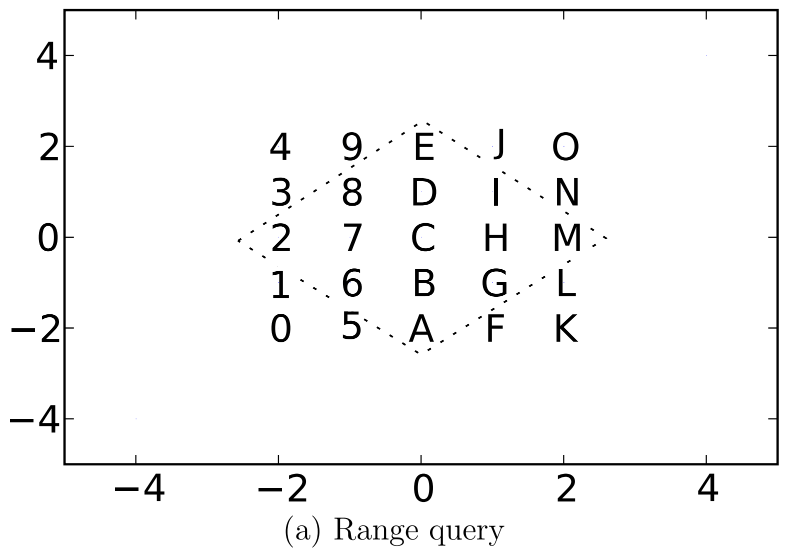
In 2-D case, the transformation is easy and involves simple axis alignment. The transformation is challenging in the case where the number of dimensions is greater than two. It is generally difficult to align the query space with the axes without blowing up the number of dimensions, which is not desirable for building efficient indexes. We tackle this problem by transforming two dimensional projections of the space. As an artifact of this, the bounding box no longer models the range query correctly. We propose a novel pruning method to alleviate this problem.

3.1 2-D Transformations

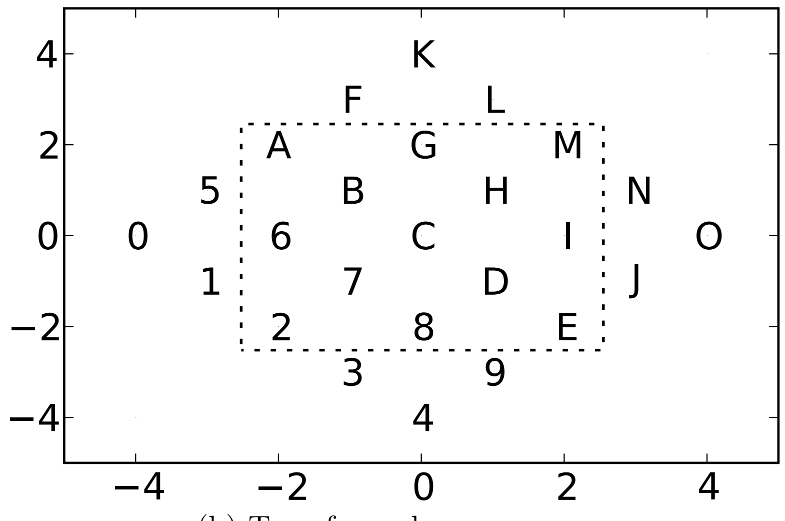
Mapping range queries into box queries requires transformations of both data space and user queries. The space transformation is performed offline on the data, before building the index. For simplicity, we call a database built with the transformed data a "transformed database". The query transformation is performed online on each query. The transformation needs to satisfy the property that the result of the transformed query over the transformed database is equal to, or at least a superset of, the result of the original query over the original database. This requirement ensures that the transformed query does not miss any data objects which would have been retrieved by the original query. When the two query results are equal (i.e. there are no false positives or false negatives), we call such transformations *precise transformations*; otherwise, we call them *approximate transformations*. Approximate transformation may introduce false positives (*i.e.*, the points that do not satisfy the original query but do satisfy the transformed query) or false negatives (*i.e.*, the points that are in the original query but are not in the transformed query). Precise transformations have neither false positives nor false negatives. We now formally present the proposed transformations. It will be shown later that our transformations are precise.

3.1.1 Transformation Function

In this section, we present the transformation function from range queries to box queries for 2-D spaces. We also highlight some of the important properties of the proposed transformation which are important for extending the transformation in higher dimensional spaces.



(a) Range query



(b) Transformed range query

Figure 3.1: Example of the query transformation

3.1.1.1 Space Transformation:

Intuitive basis for the 2-D space transformation is illustrated in the example in Figure 3.1a. Note that the edges of the range query follow the line vectors $\langle 1,\ 1\rangle$ and $\langle -1,\ 1\rangle$. If we adjust the query space's axes to align with these vectors $\langle 1,\ 1\rangle$ and $\langle 1,\ -1\rangle$ instead of the units vectors $\langle 1,\ 0\rangle$ and $\langle 0,\ 1\rangle$, our query space become the space shown in Figure 3.1b. It is interesting to see that in the transformed space, the minimal bounding box $b@([-2,2],[-2,2])$

precisely defines our original range query in Figure 3.1a.

The space transformation function can be designed in a number of ways. The most direct way is to use a space rotation by and angle of 45°. This gives us the following transformation function:

$$T(x,y) = (\frac{x-y}{\sqrt{2}}, \frac{x+y}{\sqrt{2}}) \tag{3.1}$$

However, we can achieve the same effect of mapping a range query on to the box query using a more elegant transformation. Our transformation from range queries to box queries is accomplished by mapping each point (x, y) in the original 2-D space to the point $(x+y, x-y)$ in the transformed space, which is essentially a change of axis as shown in figure 3.1.

Formally, our transformation function $T\colon \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is defined as,

$$T(x,y) = (x+y, x-y) \tag{3.2}$$

And the inverse transformation function $T^{-1}\colon \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is defined as,

$$T^{-1}(x,y) = (\frac{x+y}{2}, \frac{x-y}{2}) \tag{3.3}$$

This function essentially changes the two axes so that they align perfectly with the geometric faces of range queries in the original space. It can be seen that the function defined in equation 3.2 is computationally easier than the one in equation 3.1. Further, the transformation is an inverse of itself (with a scaling factor). By such transformations, a range query in the original space based on L_1 distance becomes a box query in the transformed space. For any 2-D database **D**, which is a set of 2-D points in the original space, the transformed database $T(\mathbf{D})$ is defined by

$$T(\mathbf{D}) = \{T(x, y) \mid (x, y) \in \mathbf{D}\}$$

Note that we do not need to store $T(\mathbf{D})$ as a separate database; rather, we build an index for $T(\mathbf{D})$, which points back to the original points in $\mathbf{D}$.

3.1.1.2 Query Transformation:

Mathematically, $r@(a, b)$ denotes the set of all the points that are within range r based on L_1 distance from point (a, b). Geometrically, all the points in $r@(a, b)$ form a diamond with four end points: $(a + r, b)$, $(a, b + r)$, $(a - r, b)$, $(a, b - r)$. On the other hand, all points in $b@([a_1, b_1], [a_2, b_2])$ form a rectangle with four end points: (a_1, b_1), (a_2, b_1), (a_2, b_2), (a_1, b_2).

The space transformation transforms the four corners of the range query $r@(a, b)$ to the four points, $(a + r + b, a + r - b)$, $(a - r + b, a - r - b)$, $(a + r + b, a - r - b)$, $(a - r + b, a + r - b)$, respectively. The square defined by these transformed points is precisely the space corresponding to the range query in transformed space. This square is essentially the representation of a box query $b@([a+b-r, a+b+r], [a-b-r, a-b+r])$ in the transformed space. Geometrically, our 2-D transformation from range queries to box queries converts a diamond in the original space to a square in the transformed space. For example, in figure 3.1, range query $2@(0, 0)$ in the original space is equivalent to the box query $b@([-2, 2], [-2, 2])$ in the transformed space.

3.1.2 Transformation Properties

We now present several important properties of our transformation function T defined in formula 3.2. These properties are particularly useful in extending 2-D transformation for higher dimensions.

3.1.2.1 Precision Property

Theorem 3.1.1 shows that the proposed transformation from an L_1 range query to a box query is precise.

Theorem 3.1.1. *For any point $(x, y) \in D$, (x, y) satisfies the range query $r@(a, b)$ if and only if (iff) $T(x, y)$ satisfies the box query $b@([a + b - r, a + b + r], [a - b - r, a - b + r])$.*

Proof 3.1.1. *Our proof is based on the fact that for any two numbers u and v, $|u| + |v| \leq r$ iff $|u + v| \leq r$ and $|u - v| \leq r$. This fact can be easily proved by assuming $u > v$ without loss of generality and then considering the following three cases: (1) $u > v > 0$, (2) $u > 0 > v$, and (3) $0 > u > v$. We omit the proof of this fact.*

Based on this fact, $|x-a|+|y-b| \leq r$ holds iff both $|(x+y)-(a+b)| = |(x-a)+(y-b)| \leq r$ and $|(x - y) - (a - b)| = |(x - a) - (y - b)| \leq r$ hold. Note that (x, y) satisfies the range query $r@(a, b)$ iff $|x - a| + |y - b| \leq r$ holds, and $T(x, y)$ satisfies the box query $b@([a+b-r, a+b+r], [a-b-r, a-b+r])$ iff $|(x+y)-(a+b)| \leq r$ and $|(x-y)-(a-b)| \leq r$ holds.

3.1.2.2 Distance Property

The proposed transformation function T does not preserve L_1 distance, even though it is precise. Consider two points $p = (1,\ 1)$ and $q = (1.5,\ 0)$. Distance of these points from origin is 2 units and 1.5 units respectively. Using the transformation, $T(p) = (2,\ 0)$ and $T(q) = (1.5,\ 1.5)$. As origin is unaffected by the transformation, the distances of transformed points from the (transformed) origin are 2 units and 3 units respectively. It can be seen that neither the distance nor the relative ordering of points is preserved.

Theorem 3.1.2. *For any two points* (x_1, y_1) *and* (x_2, y_2)*,* $L_1(T(x_1, y_1), T(x_2, y_2)) = L_1((x_1, y_1), (x_2, y_2)) + ||x_1 - x_2| - |y_1 - y_2||$.

Proof 3.1.2. *Our proof is based on the fact that for any two numbers* u *and* v*,* $|u + v| + |u - v| = |u| + |v| + ||u| - |v||$*. This fact can be proved easily proved by assuming* $u > v$ *without loss of generality and then considering the following three cases: (1)* $u > v > 0$*, (2)* $u > 0 > v$*, and (3)* $0 > u > v$*. We omit the proof of this fact.*

Based on this fact, we have $L_1(T(x_1, y_1), T(x_2, y_2)) = L_1((x_1+y_1, x_1-y_1), (x_2+y_2, x_2-y_2)) = |(x_1+y_1)-(x_2+y_2)|+|(x_1-y_1)-(x_2-y_2)| = |(x_1-x_2)+(y_1-y_2)|+|(x_1-x_2)-(y_1-y_2)|$ $= |x_1-x_2|+|y_1-y_2|+||x_1-x_2|-|y_1-y_2|| = L_1((x_1, y_1)-(x_2, y_2))+||x_1-x_2|-|y_1-y_2||$.

We can prove a similar property for T^{-1}.

Corollary 3.1.1. *For any two points* (x_1, y_1) *and* (x_2, y_2)*,* $L_1(T^{-1}((x_1, y_1)), T^{-1}((x_2, y_2))) = (L_1((x_1, y_1), (x_2, y_2)) + ||x_1 - x_2| - |y_1 - y_2||)/2$.

3.1.2.3 Inequality Property:

Although transformation function T does not preserve L_1 distance, Theorem 3.1.3 shows an important special case where the transformation function T preserves distance inequality.

Theorem 3.1.3. *Given a point* (x_1, y_1)*, an MBR represented as a rectangle* $\boldsymbol{B}$*, and a point* (x_2, y_2) *on the edge of the rectangle, if among all the points in* $\boldsymbol{B}$*,* (x_2, y_2) *is the point*

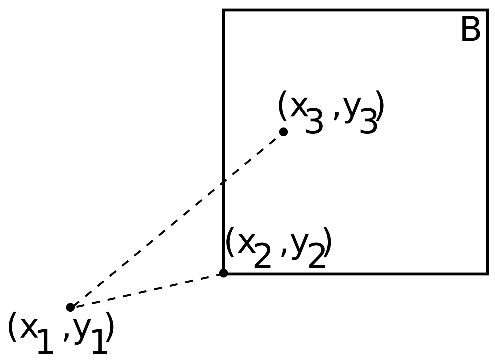


Figure 3.2: Illustration for Theorem 3.1.3

that is closest to (x_1, y_1), then $T((x_2, y_2))$ is the closest point in $T(\boldsymbol{B})$ to $T((x_1, y_1))$ and $T^{-1}((x_2, y_2))$ is the closest point in $T^{-1}(\boldsymbol{B})$ to $T^{-1}((x_1, y_1))$. Here, $T(\mathbf{B})$ and $(T^{-1})(\boldsymbol{B})$ are the sets obtained by transforming all the points in $\mathbf{B}$ using the transformations T and T^{-1} respectively.

Proof 3.1.3. *Our proof is based on the fact that for any four non-negative numbers u, v, w, and z, if $u+v \leq w+z$, $u \leq w$, and $v < z$, then $u+v+|u-v| \leq w+z+|w-z|$. This fact can be easily proved by considering the following three cases: (1) $u \leq w \leq v \leq z$, $u \leq v \leq w \leq z$, and $u \leq v \leq z \leq w$. We omit the proof of this fact.*

For any two points (x_1, y_1) and (x_2, y_2), we use $L_1((x_1, y_1), (x_2, y_2))$ to denote their L_1 distance. Considering any point (x_3, y_3) in the rectangle, because (x_2, y_2) is closer to (x_1, y_1) than (x_3, y_3), we have $L_1((x_2, y_2), (x_1, y_1)) \leq L_1((x_3, y_3), (x_1, y_1))$, $|x_2 - x_1| \leq |x_3 - x_1|$ and $|y_2 - y_1| \leq |y_3 - y_1|$. Now we need to prove $L_1(T((x_2, y_2)), T((x_1, y_1))) \leq L_1(T((x_3, y_3)) - T((x_1, y_1)))$. By Theorem 3.1.2, we have $L_1(T((x_2, y_2)), T((x_1, y_1))) = L_1((x_1, y_1), (x_2, y_2)) + ||x_1 - x_2| - |y_1 - y_2|| = |x_1 - x_2| + |y_1 - y_2| + ||x_1 - x_2| - |y_1 - y_2||$ and $L_1(T((x_3, y_3)), T((x_1, y_1))) = L_1((x_1, y_1), (x_3, y_3)) + ||x_1 - x_3| - |y_1 - y_3|| = |x_1 - x_3| + |y_1 - y_3| + ||x_1 - x_3| - |y_1 - y_3||$. By the above fact, we have $|x_1 - x_2| + |y_1 - y_2| + ||x_1 - x_2| - |y_1 - y_2|| \leq |x_1 - x_3| + |y_1 - y_3| + ||x_1 - x_3| - |y_1 - y_3||$.

Figure 3.2 shows an example scenario for points (x_1, y_1), (x_2, y_2), (x_3, y_3) and the rectangle.

3.2 Multi-dimensional Transformations

For number of dimensions $n = 2$, range queries can be precisely transformed into box queries. However, to the best of our knowledge, for $n > 2$ we have not found any heuristics in the literature on precise transformations. We conjecture that such a precise transformation may not exist. Intuitive justification for this conjecture can be given by considering the number of vertices and number of faces in the query space. Range query in L_1 has $2n$ vertices and 2^n faces (In 3-D, 6 vertices and 8 faces). A box query on the other hand, has 2^n vertices and $2n$ faces (In 3-D, 8 vertices and 6 faces). In fact, these two query spaces are duals of each other in the sense that vertex in range query space is a face in box query space and vice versa. Hence, it may not be possible to map range query space onto the box query space using simple affine transformations.

In this section, we first use the 2-D transformation of section 3.1 to define an approximate transformation called disjoint planar rotations (DPR). Since the transformation is not precise, the basic high dimensional box query as defined in equation 1.5 results in a lot of false positives which may negatively affect the performance. Hence, we develop a new type of box query that uses the range value from the original range query to prune the false positives in the transformed box query while preventing the occurrence of false negatives.

3.2.1 Disjoint Planar Rotations

DPR is a transformation that is derived from our two dimensional transformation function. We transform the n dimensional space via this technique by transforming disjoint planes in the database. For example, a four dimensional point (x, y, z, w) can be transformed into $(T(x, y), T(z, w))$. That is, this transformation can be visualized as a rotation of each of the 2-D disjoint planes in the database's space.

More formally, we define a n dimensional transformation $T^n(p)$ of a point $p = (p_1, p_2, \ldots, p_n)$, as follows:

$$T^n(p) = \begin{cases} (T(p_1, p_2), \ldots, T(p_{n-1},\ p_n)) & \text{when} n \text{ iseven} \\ (T(p_1, p_2), \ldots, T(p_{d-2}, p_{n-1}),\ p_n) & \text{when} n \text{isodd} \end{cases} \quad (3.4)$$

Note that when n is odd, we choose to preserve the last dimension because in 1-dimensional space, range query and box query are one and the same and they differ only in the representation (e.g. $r@(x) \Leftrightarrow b@([x-r, x+r])$), which obviates the need for an explicit transformation.

3.2.2 Pruning Box Query

Our modification to box query (which we call Pruning Box Query - PBQ) is based on the observation that if we can estimate distance between the query center and an MBR of the index tree, we can prune the branches of the tree that do not contain any true positives. We first prove the result proposed in theorem 3.1.3 for $n-$dimensional data.

Theorem 3.2.1. *Given a point c and an MBR $\mathbf{B}$. If p is the closest point in $\mathbf{B}$ to c, then $T^n(p)$ is the closest point in $T^n(\mathbf{B})$ to $T^n(c)$ and $(T^{-1})^n(p)$ is the closest point in $(T^{-1})^n(\boldsymbol{B})$ to $(T^{-1})^n(c)$. Here, $T^n(\mathbf{B})$ and $(T^{-1})^n(\boldsymbol{B})$ are the sets obtained by transforming all the points in $\mathbf{B}$ using the transformations T^n and $(T^{-1})^n$ respectively.*

Proof 3.2.1. *The proof is similar to that of theorem 3.1.3 and is omitted.*

Based on the theorem we propose following heuristic to eliminate all the false positives:

Heuristic: If an MBR $\mathbf{M}$ overlaps with the query box, we find the closest point $T^n(p)$ in M to query center $T^n(c)$. Using the inverse transformation we then calculate distance between p and c in the original space; if it is greater than the query range then the MBR is pruned. Else it is recursively searched. Once we reach the leaf level nodes, for each record $T^n(q)$ in the node, we calculate the L_1 distance between q and c. If this distance is less than the query radius, q is added in the resultset.

This approach not only eliminates all the false positives but it also provides more efficient query execution. It is important to note that we do not miss any data record with this pruning strategy. Theorem 3.2.2 states this.

Theorem 3.2.2. *For every point p that satisfies the range query $r@(c_1, c_2 \ldots c_n)$, p is also contained in the result of the PBQ.*

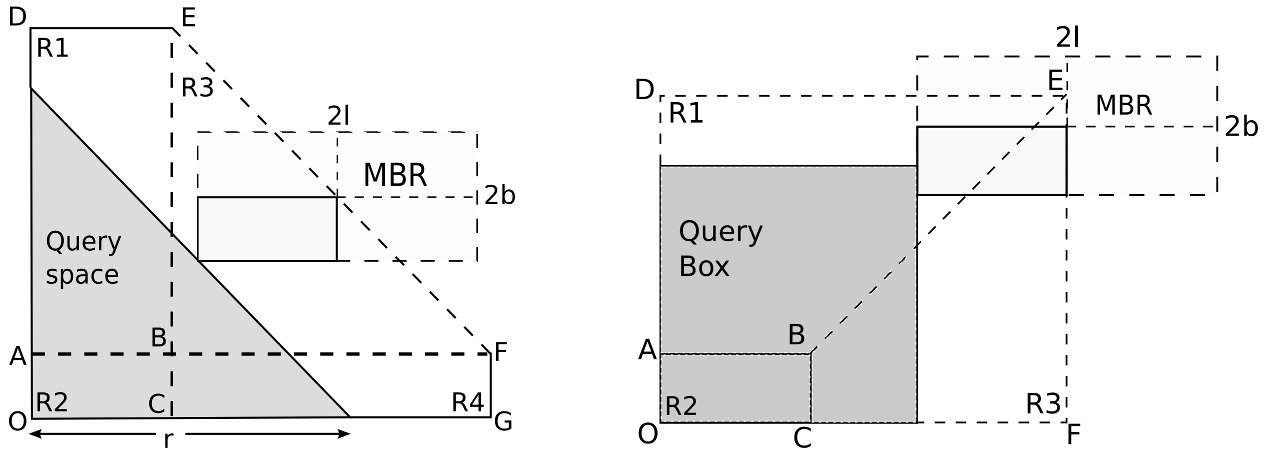
Proof 3.2.2. *Let $c = (c_1, c_2 \ldots c_n)$. Further, Let, if possible, there be a point p such that, $p \in r@(c_1, c_2 \ldots c_n)$ but p' is not contained in the pruning box query, where $p' = T^n(p)$. This is possible, only if the MBR $\mathbf{M}$ containing p' was pruned by the box query at some point, i.e. the estimated distance between M and c' ($c' = T^n(c)$) was greater than r. Let $u' = T^d(u)$ be the closest point in $\mathbf{M}$ to c'. This implies that while u' was the closest point to c' in the transformed domain, u was not the closest point to c in the original data domain. This contradicts theorem 3.2.1. Hence, such a point p cannot exist. In other words, resultset returned by the PBQ contains all the points from the one returned by the original range query.*

3.3 Theoretical Analysis

Using DPR and pruning box queries improves the performance of indexed queries because the transformation aligns the index's minimum bounding boxes' faces with faces of the query. In this section we provide a detailed analysis of the range query and the PBQ performances. We first present it for 2-dimensional queries and then generalize it for higher dimensions.

3.3.1 Model Basics

Without loss of generality, we fix the size of all MBRs so that we can calculate the area of MBR centroids whose corresponding MBRs intersect with a query. From this area, we can calculate the probability that an MBR of a certain size will intersect with a query. For example, for 2D query spaces, we fix an MBRs length to be $2l$ and breadth to be $2b$. We must calculate probability that a random MBR of certain size intersects the query space (a diamond in case of the range query and a square for the PBQ). We analyze only one quadrant of the 2-D plane. Analysis for other quadrants is similar and is omitted.



(a) Range query (b) Transformed box query

Figure 3.3: The MBR intersection areas in a quadrant

Figure 3.3 shows the space in which an MBR of size $2l \times 2b$ must lie in order to intersect with the query space. We can see from this visualization that the query faces align with the MBR faces after transformation, and we conjecture that this alignment improves query performance for two reasons: MBRs are less likely to intersect with the PBQ than the range query, and MBRs that do intersect with PBQ have a higher likelihood of containing a point within the query than MBRs that intersect the range query.

3.3.2 Analysis of Range Query

To calculate the probability of intersection for a range query $P_{R(r,l,b)}$ with an MBR, we determine the area in which the centroid of an MBR with length $2l$ and breadth $2b$ must lie for it to intersect with the query. As shown in Figure 3.3(a), the intersection space can be divided into four regions $R_1(\square ADEB), R_2(\square OABC), R_3(\triangle BEF)$ and $R_4(\square CBFG)$. The area of the intersection space can then be calculated as,

$$\begin{aligned} A_{R(r,l,b)} =& \text{Area of R1, R2, R3, and R4} \\ =& \int_0^r b \ dx + lb \\ &+ \int_0^{\frac{r}{\sqrt{2}}} 2x \ dx + \int_0^r l \ dx \\ =& \frac{r^2}{2} + lb + r(l+b) \end{aligned} \tag{3.5}$$

Hence, given the area of the data space, A, the probability that an MBR will overlap with a range query is,

$$P_{R(r,l,b)} = \frac{A_{R(r,l,b)}}{A} \tag{3.6}$$

3.3.3 Analysis of Box Query

To calculate the probability $P_{B(r,l,b)}$ of intersection of a PBQ with an MBR, we determine the area in which the centroid of an MBR with length $2l$ and breadth $2b$ must lie for it to intersect with the query. As shown in Figure 3.3(b), the intersection space is an extended box query with length of $2(l+r)$ and breadth of $2(b+r)$. Note that the space has dilated by a factor of 2 due to the transformation. Hence, the total area of the space in the transformed domain is $2A$. We divide the space into three regions $R_1(\square ADEB)$, $R_2(\square OABC)$ and $R_3(\square CBEF)$. The area of the intersection space can then be calculated as,

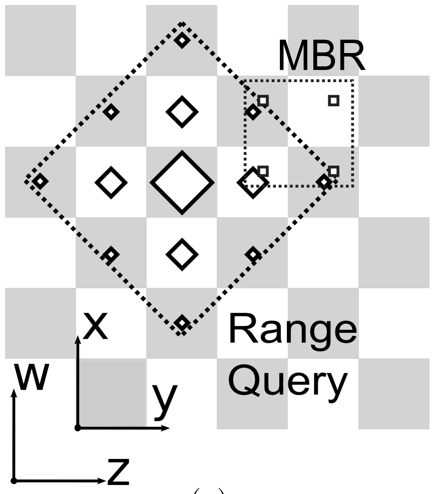
$$\begin{aligned} A_{B(r,l,b)} =& \text{Area of R1, R2, and R3} \\ =& \int_0^r (l+x)\ dx + lb \\ &+ \int_0^r (b+x)\ dx \\ =& r^2 + lb + r(l+b) \end{aligned} \tag{3.7}$$

Hence, given a the area of the data space, $2A$, the probability that a random MBR intersects the transformed range query is,

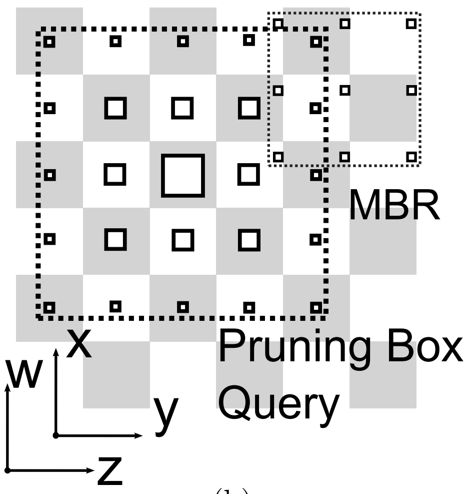
$$P_{B(r,l,b)} = \frac{A_{B(r,l,b)}}{2A} \tag{3.8}$$

3.3.4 Hyper-dimensional Queries

Figure 3.4 represents the four dimensional space as a two dimensional grid of two dimensional slices through the four dimensional space. In this case, the grid is a coarse grain visualization of the effect of the wz plane on the xy planes so each panel in the wz represents the xy plane



(a)



(b)

Figure 3.4: Visualizations of range and pruning box queries relationship with MBRs in hyperspace

that is fixed at the wz panel's coordinate. While this visualization is coarse grained in that it does not show every point in the range query, it illustrates how DPR transforms the range query into a PBQ as shown in Figure 3.4(b), which shows Figure 3.4(a)'s range query as a PBQ in a DPR transformed space. Note that this visualization can be generalized to visualize any hyperspace as a nested series of two dimensional grids. The above equations can be generalized to any even number of dimensions via a density function. We use this nesting concept to find the area of centroids for any even dimensional query. As we move away from the center of the query in xy plane, density of points (or query space) in wz plane decreases. We define density function as the area in which center of an MBR must lie in order to intersect with query space. The n-dimensional density function for range query is denoted as $A_{R(r,W,n)}$ and that for the pruning box query is denoted as $A_{B(r,W,n)}$.

Consider the MBR in Figure 3.4(a); we first examine the intersections of the wz projections of the MBR and query, which is shown by the dotted lines. Note that if these projections did not intersect there would be no intersection of query and MBR; however, since there is an intersection we can determine if query and MBR do intersect by looking for an intersection in the xy projection that is closest to the origin of the range query.

Given a hyper-rectangle with widths $W = \langle w_1, \cdots, w_n \rangle$, the density function for the range query is recursively defined as,

$$
\begin{aligned}
A_{R(r,W,0)} =& 1 \\
A_{R(r,W,n)} =& \int_0^r A_{R(r-x,W,n-2)} w_n \, dx \\
& + A_{R(r,W,n-2)} w_{n-1} w_n \\
& + \int_0^{\frac{r}{\sqrt{2}}} A_{R(r-x\sqrt{2},W,n-2)} 2x \, dx \\
& + \int_0^r A_{R(r-x,W,n-2)} w_{n-1} \, dx \qquad (3.9) \\
P_{R(r,W,n)} =& \frac{A_{R(r,W,n)}}{A} \qquad (3.10)
\end{aligned}
$$

in the original space and

$$
\begin{aligned}
A_{B(r,W,n)} =& 1 \\
A_{B(r,W,n)} =& \int_0^r A_{B(r-x,W,n-2)} (w_{n-1} + x) \, dx \\
& + A_{B(r,W,n-2)} w_{n-1} w_n \\
& + \int_0^r A_{B(r-x,W,n-2)} (w_n + x) \, dx \qquad (3.11) \\
P_{B(r,W,n)} =& \frac{A_{B(r,W,n)}}{2^{n/2} A} \qquad (3.12)
\end{aligned}
$$

in the transformed space. Again, the space dilation is responsible for the factor $2^{n/2}$ in the equation 3.12. It can be seen that area analyses for two dimensional cases are in fact special cases of equations 3.9 through 3.11

It can be shown that the density function for the PBQ is less than the density function for the range query, when we increase the number of dimensions. Hence, the range query has a larger number of valid centroids that intersect it than the pruning box query.

It should be noted that this analysis does not provide an exact modeling of the actual trees because, MBRs are not completely random in any index tree (due to the use of heuristics), and the analysis holds if all the MBRs have the same sides which is generally not the case.

3.4 Experimental results with R*-Tree

In this section we present the results of applying the proposed transformation on various databases. Effectiveness of the proposed transformation is measured by comparing the IO cost (i.e. number of index page accesses) for the proposed pruning box queries with that of range queries.

For performance comparison purposes, we create R*-tree index for the range query in the original space and R*-tree index for the pruning box query in the transformed space. Later in the chapter we highlight some of the shortcomings of R*-Tree which can be overcome by using packed R-Tree. We demonstrate with comprehensive experiments that the proposed transformation is equally effective with packed R-Tree as well.

Uniformly distributed synthetic databases are used for the experiments. Data records are normalized to unit (hyper)cube. Page size of 4K bytes was used for the index nodes. All results presented here are based on averaging the I/O of one hundred random queries. All the experiments were run on AMD Opteron 2.2.GHz systems running GNU/Linux. The labels used for various methods in the figures and tables are as follows: RQ - traditional range query on R*-Tree, PBQ - Pruning Box Query on R*-Tree.

3.4.1 2-D transformations

As explained earlier, transformation in 2-dimensional databases is precise, i.e., the transformed query does not lose any useful results nor does it gather any unwanted results. Here, we present effects of increasing database size and query radius on performance gain obtained by the transformation.

3.4.1.1 Effect of database size

Figure 3.5 shows the effect of increasing database size on the number of page accesses. As seen from the figure, as the database size increases, number of page accesses for both range and pruning box queries increases, as expected. However, rate of increase for range query is higher than that of the pruning box query. The performance improvement increases with increasing database size. The relatively small improvement is consistent with our analysis in section 3.3.

3.4.1.2 Effect of query ranges

We experimented with various query ranges keeping database size constant (50 million records). The performance comparison of pruning box query with range query is shown in figure 3.6. Ranges in the figure are a normalized distances. It can be seen from the figure that pruning-box queries perform consistently better than range queries, however, percentage improvement is not very significant.

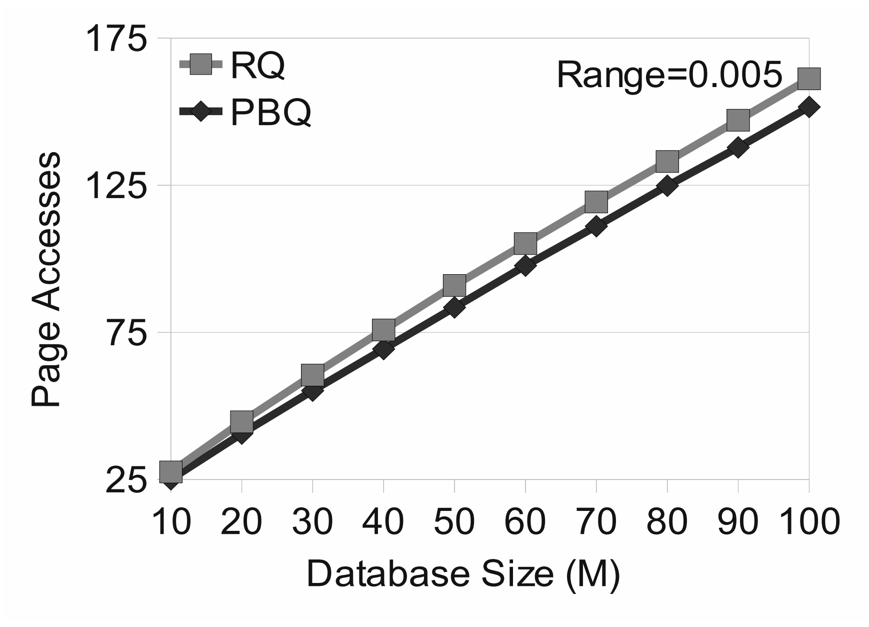


Figure 3.5: Effect on database size on page accesses

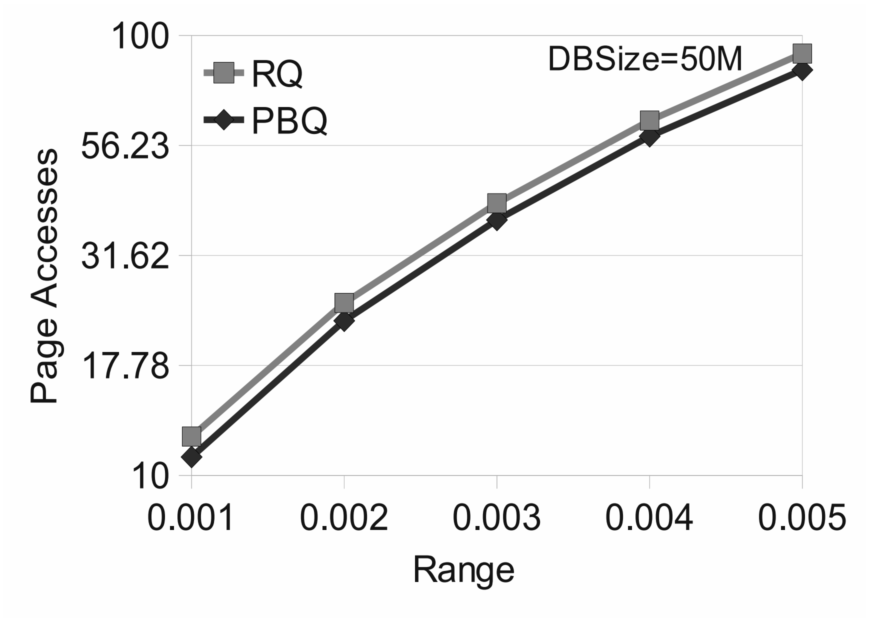


Figure 3.6: Effect of query range on page accesses

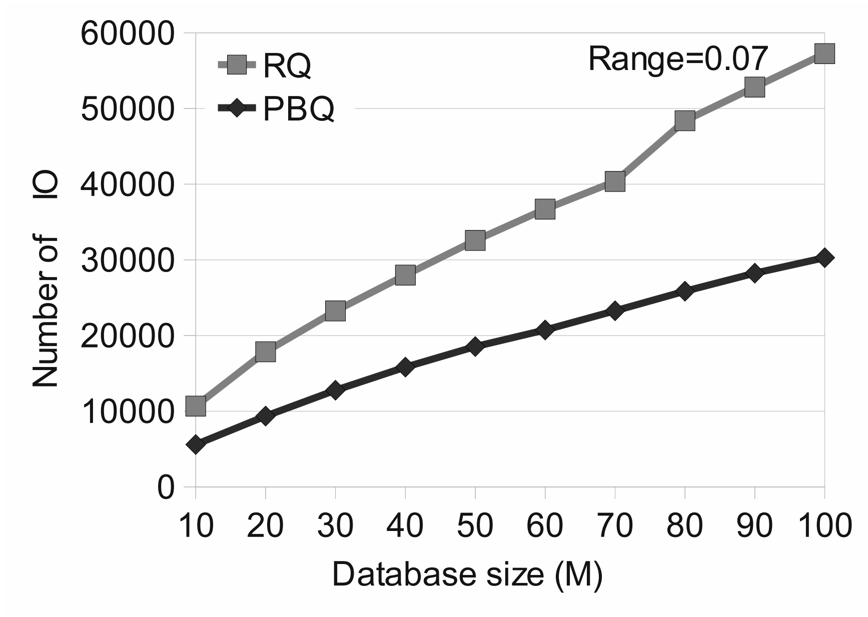


Figure 3.7: Effect of database size on page accesses

3.4.2 Higher dimensional transformation

The main challenge in transforming high dimensional queries is that the DPR transformation tends to retrieve a lot of false positives. We use the pruning box query to eliminate false positives and reduce the number of page accesses for execution of the query. In the following subsections, we present the results for 10-dimensional uniformly distributed synthetic data.

3.4.2.1 Effect of database size

Figure 3.7 shows effect of database size on the query cost. We used a database with 10-dimensional vectors. Query range was kept constant. As can be seen from the figure, as the database size increases, cost for both range and box queries increases, as expected, but the rate of increase is much slower for pruning box queries than for range queries. We get more than 40% reduction in the cost for a database size of 100 million vectors.

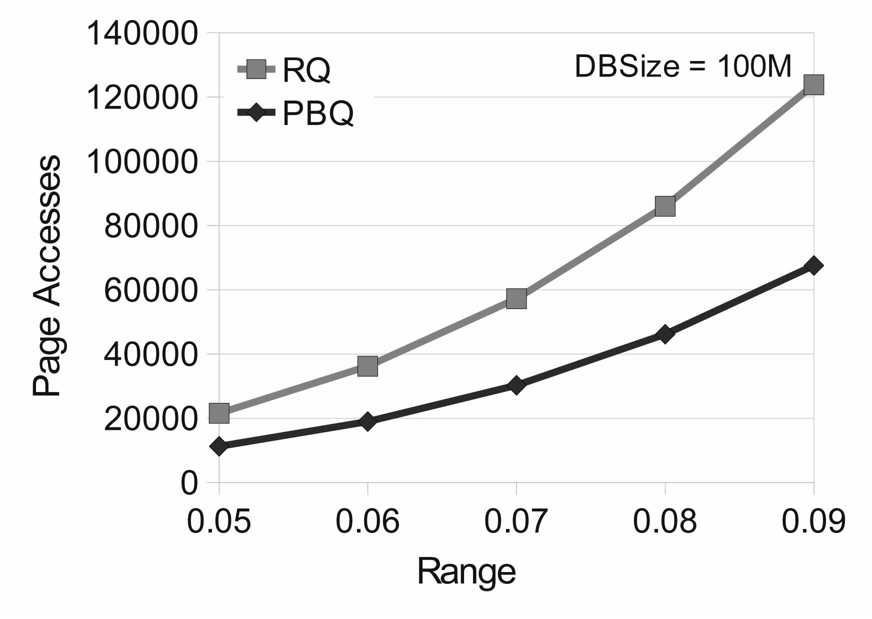


Figure 3.8: Effect of range on page accesses

3.4.2.2 Effect of query ranges

Figure 3.8 gives the comparative performance of range queries versus pruning box queries with increasing query range. As seen from the figure, performance of pruning box queries is consistently better than range queries, and the performance difference gets wider with increasing query ranges. This is because hyper-diamonds of the range queries tend to intersect more with the bounding boxes than the pruning box queries.

3.5 Effect of the index structure

We have observed that the performance improvement due to transformation is sensitive to the structure of the R*-tree being built. One of the primary reasons for this is the fact that R*-Tree (and many dynamic index structures) are very sensitive to the order in which data is being inserted. Although R*-Tree makes some attempt to improve the index through periodic reinsertion of some of the data points, it is unable to nullify the effect of insertion order completely. This sensitivity makes it difficult to assess improvement due to transformation. As we are building two separate indexes (one in transformed space and one in original space), it is possible that one of the index has a favorable order while the other does not. In other words, improvement may result from combined effect of transformation and insertion order.

We performed extensive empirical analysis to study the effects of insertion orders on the performance of R*-tree. We also studies R*-tree performance for several different databases all coming from the same distribution. Tables 3.1 and 3.2 summarize our findings. For all the experiments, database size was fixed at 10 million records. We used 10 different insertion orders within the same database for the first experiment and 10 different databases (all with uniform distribution) for the second. Average number of page accesses and standard deviation of the number of page accesses was calculated. It can be seen from these tables that there exists a large variance in query performance of R*-Tree. There are cases where performance improvement is observed to be in excess of 40%. What this analysis shows is that for a given database, it may be difficult to isolate the performance improvement due to transformation from the positive or negative effects of insertion order and statistical properties of data.

Table 3.1: Mean and Std. deviation of # page accesses for various insertion orders

Query type	**# Dimensions**	**Mean IO**	**Std. Deviation**
RQ	2	27.67	0.21
	10	7678.05	1626.54
PBQ	2	25. 09	0.46
	10	7609.86	1465.36

The inherent dependence of R*-Tree on insertion order and statistical properties of the database may make it difficult to effectively measure the performance improvement due to

Table 3.2: Mean and Std. deviation of # page accesses for various databases

Query type	# Dimensions	Mean IO	Std. Deviation
RQ	2	27.77	0.35
	10	7220.27	1702.33
PBQ	2	24.88	0.35
	10	6509.47	1375.35

transformation. We apply the proposed transformation on a static index called packed R-Tree [41]. Packed R-Tree (and any static index) assumes that all the data is available at the time of indexing. This allows for better tuning of the index. Packed R-Tree sorts the data using Hilbert ordering before inserting. Further, it tries to put as many data records in a node as possible. This creates more compact R-Tree in which all the records in a leaf node are spatially close to each other. Due to pre-sorting, the order in which the data comes in does not matter, it is always indexed in the same order. Further, Packed R-Tree also guarantees (almost) 100% space utilization which considerably reduces the total number disk pages required for the index.

Similar to the R*-Tree we measured mean and variance of number of page accesses for 10 different databases. Table 3.3 shows results of these experiments. It can be seen that standard deviation in the number of page accesses for a packed R-Tree is very small highlighting its stability. At the same time performance gain in terms of page accesses is significant even for higher dimensions.

Table 3.3: Mean and Std. deviation of # page accesses for various databases indexed using packed R-tree

Query type	# Dimensions	Mean IO	Std. Deviation
RQ	2	29.21	0.14
	10	3212.96	3.28
PBQ	2	26.41	0.1
	10	2500.88	2.21

Based on these results, we decide to use Packed R-Tree for all our experiments in order to nullify any positive or negative effect of the index properties and to accurately measure performance improvement due to transformation alone.

3.6 Experimental results using Packed R-Tree

In this section we present the results of applying the proposed transformation on various databases indexed using packed R-Tree. As before, effectiveness of the transformation is measured by comparing the IO cost (i.e. number of index page accesses) for the pruning box queries with that of range queries. For performance comparison purposes, we create packed R-Tree indexes for the range query in the original space and the pruning box query in the transformed space.

A detailed analysis of effect of database sizes, number of dimensions and query radius is presented in subsequent sections. We first prove the correctness of our theoretical analysis by comparing improvement predicted by our model with the actual improvement observed experimentally.

3.6.1 Correctness of theoretical model

We define the query-performance improvement as the ratio of the number of page accesses in the transformed space to that in the original space. Based on equations 3.9 and 3.11 we can estimate expected relative improvement due to the transformation as,

$$I(r, W, n) = P_{B(r,W,n)}/P_{R(r,W,n)} \tag{3.13}$$

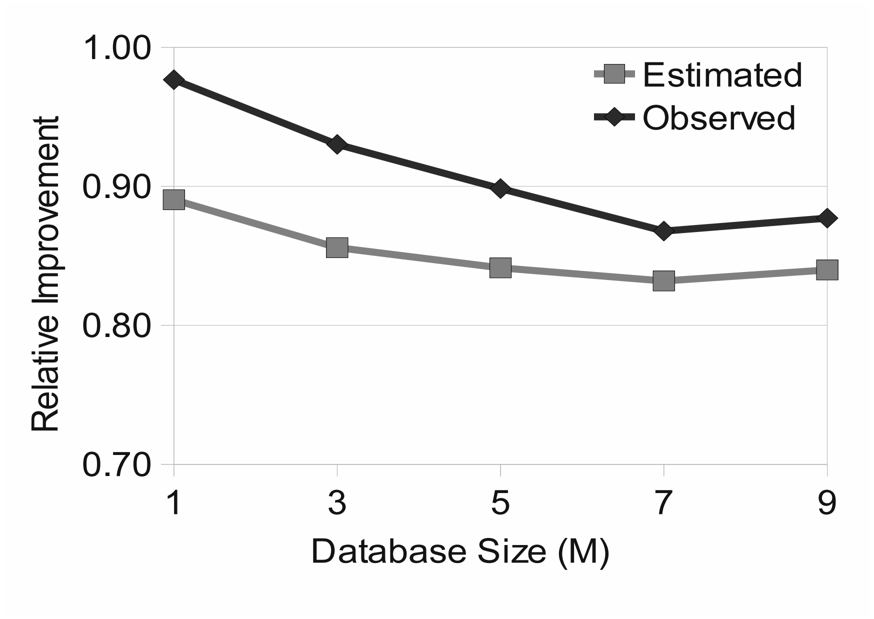


Figure 3.9: Comparison of estimated and observed improvement for 2-D data

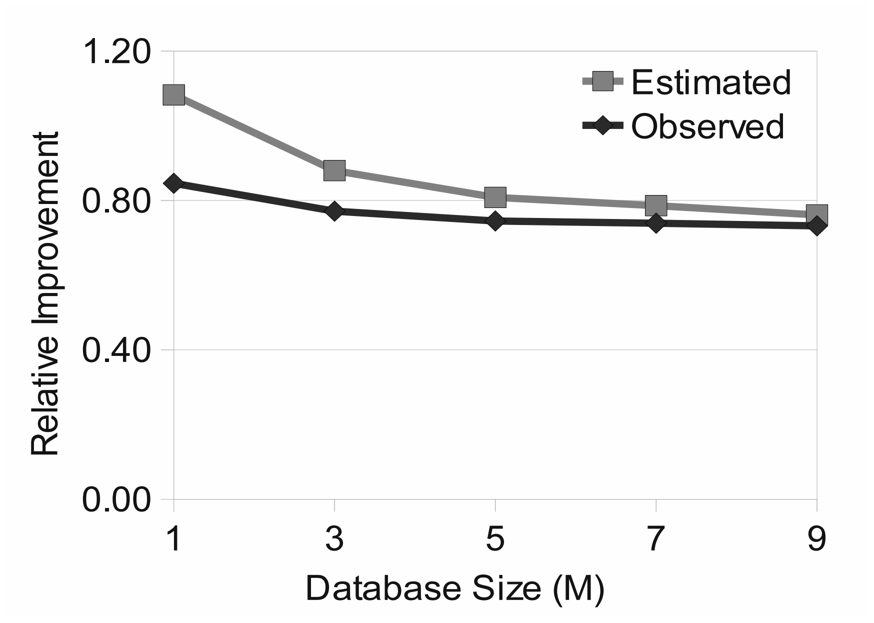


Figure 3.10: Comparison of estimated and observed improvement for 10-D data

When range r is considerably large compared to size of the MBRs, the first term in equations 3.5 and 3.7 will dominate and we won't have any significant improvement. However, in most of the real world systems, query range is usually small so that it retrieves only a few relevant records. Hence, in general, we can expect a considerable performance gain.

Figures 3.9 and 3.10 compare the estimated values of relative improvements (equation 3.13) with the observed values for a fixed query for 2 and 10-dimensional databases respectively. For ease of evaluating integrals, we assumed that all MBRs are uniform (*i.e.* $l = b$). Further, as leaf level nodes dominate the page accesses, we use only leaf level I/O for the comparison. It can be seen from the graphs that our analysis is fairly accurate especially for larger databases. When database size is small, some of our assumptions such as MBRs are uniform and they all have the same size, may not be valid. This results in slight disagreement between observed performance improvement and predicted one. However, as database size increases, the assumptions hold and we see much better agreement between the two values.

3.6.2 Avoiding empty pages

Experimentally, we can see that the reduction in area that our model predicts for pruning box queries translates to fewer page accesses in the index tree. Table 3.4 shows the performance break down at each level of the index tree for both range and pruning box queries for two and ten dimensions and database of 10 million records. The break down shows the average number of page accesses, the average number of empty page accesses, and the average number of non-empty page accesses. An *empty* page has no children that satisfy the query, while a *non-empty* page has at least one child that satisfies the query.

We observe that both range and pruning box queries have a similar number of non-empty page accesses, which corresponds in our model to the shared centroid area with both queries. The number of non-empty pages access should be similar between both query types because our transformation does not change the relative distribution of the records in the space.

It can be seen that the performance improvements are best gained by reducing the number

Table 3.4: Break down of page accesses

PBQ $n = 2, \quad r = 0.005$

Level	Total	Empty	Non-empty
Top	1.47	0.44	1.03
Middle	3.97	2.34	1.63
Bottom	20.13	3.35	16.78

RQ $n = 2, \quad r = 0.005$

Level	Total	Empty	Non-empty
Top	1.34	0.31	1.03
Middle	3.46	1.67	1.79
Bottom	23.34	5.44	17.9

PBQ $n = 10, \quad r = 0.07$

Level	Total	Empty	Non-empty
Top	1.91	0.53	1.38
Middle-h	27.76	22.31	5.45
Middle-l	269.56	255.75	13.81
Bottom	1948.66	1922.56	26.1

RQ $n = 10, \quad r = 0.07$

Level	Total	Empty	Non-empty
Top	1.95	0.56	1.39
Middle-h	34.17	29.81	4.36
Middle-l	346.53	335.74	10.79
Bottom	2690.01	2665.98	24.03

of empty page accesses. For example, when $n = 2$, there are very few empty pages, and we see a small difference in performance between queries. However, when $n = 10$, empty pages make up the majority of page accesses, and we see that the pruning box query loads approximately two third the number of empty pages than the range query, which results in a much larger performance improvement.

3.6.3 Results for 2-D transformations

Here we present the experimental results with 2-dimensional data. As before, we study effects of increasing database size and query radius.

3.6.3.1 Effect of database size

Figure 3.11 shows the effect of increasing database size on the number of page accesses. Our observations are similar to the ones noted in section 3.4.1.1. Query I/O increases with increasing database size for both the methods but the rate of increase is smaller for pruning box query. For database size of 5 million we see about 12% improvement in the number of page accesses.

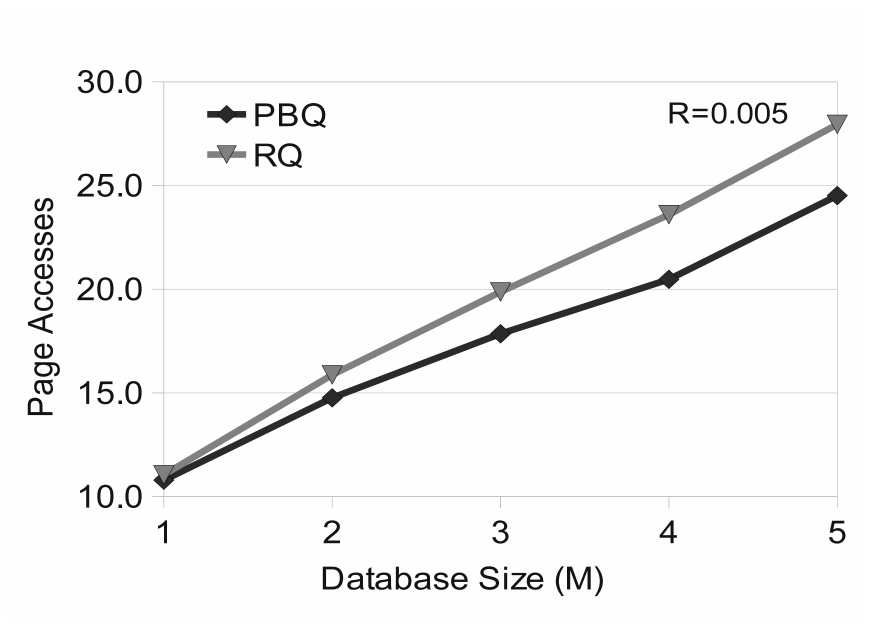


Figure 3.11: Effect on database size on page accesses

3.6.3.2 Effect of query ranges

Figure 3.12 shows the performance of the pruning box query against range query with increasing radius. It can be observed that pruning box query consistently outperforms range query in original space.

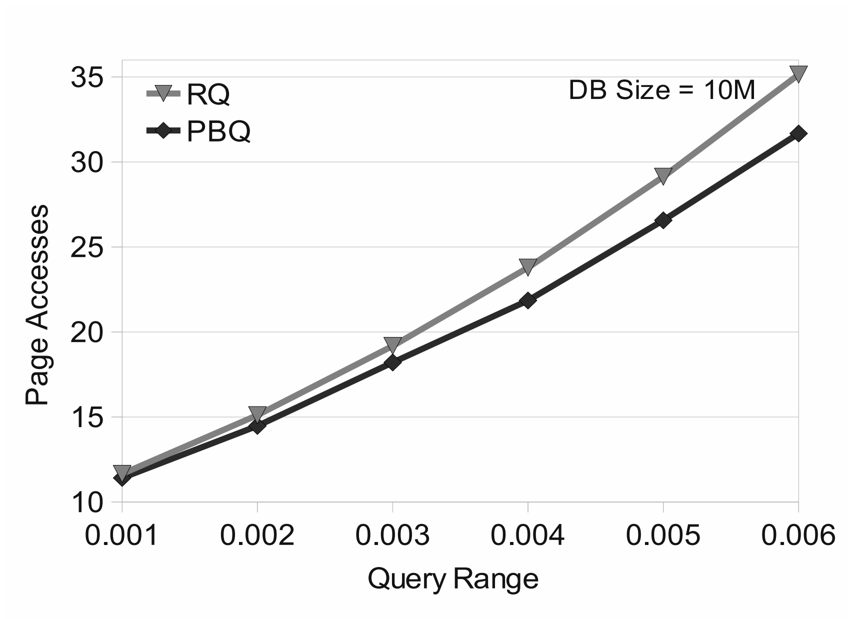


Figure 3.12: Effect of query range on page accesses

3.6.4 Higher dimensional transformation

We now present results of applying pruning box query to packed R-tree indexing 10-dimensional, uniformly distributed synthetic data.

3.6.4.1 Effect of database size

Figure 3.13 shows effect of database size on the query I/O. Significant performance gain in excess of 25% can be obtained using PBQ instead of RQ. Rate of increase in the I/O of PBQ is lesser than that for RQ which is another encouraging observation.

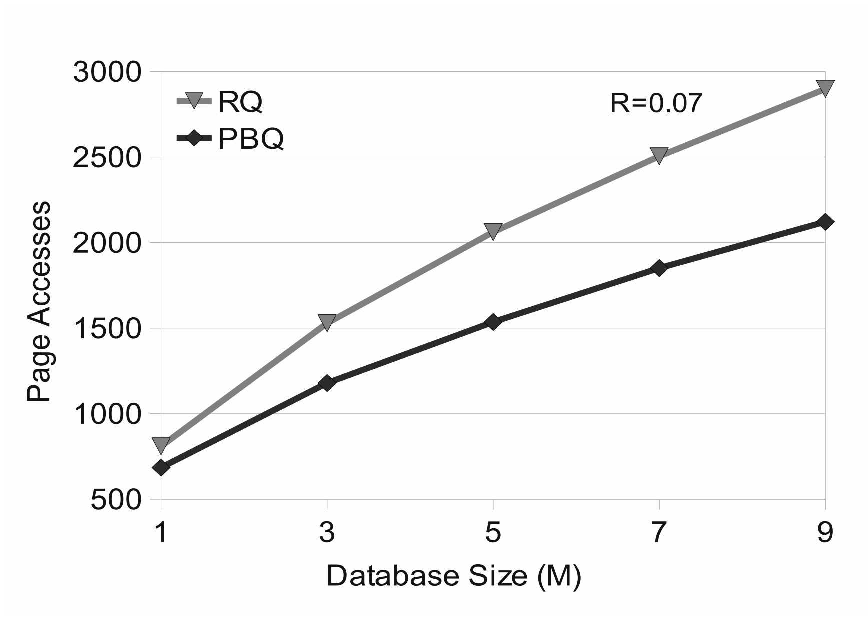


Figure 3.13: Effect of database size on page accesses

3.6.4.2 Effect of query ranges

Figure 3.14 compares performance of pruning box query with the range query for increasing query range. Note that volume of the query space increases exponentially with increasing radius. Hence rate of increase in the query I/O is also faster than linear. However, as it can be seen from the figure that the proposed method offers much better performance than the traditional range query (about 30% improvement for query radius of 0.1).

3.7 Performance results for real data

The experimental results described so far were carried out on synthetic data. In this section we show effectiveness of the proposed approach on real data. As packed R-Tree is more stable of the two index trees, we present these results only for packed R-Tree.

We use two different datasets for our experiments. First is a GIS dataset with two dimensions (co-ordinates of points obtained through GPS) and there are totally 108779

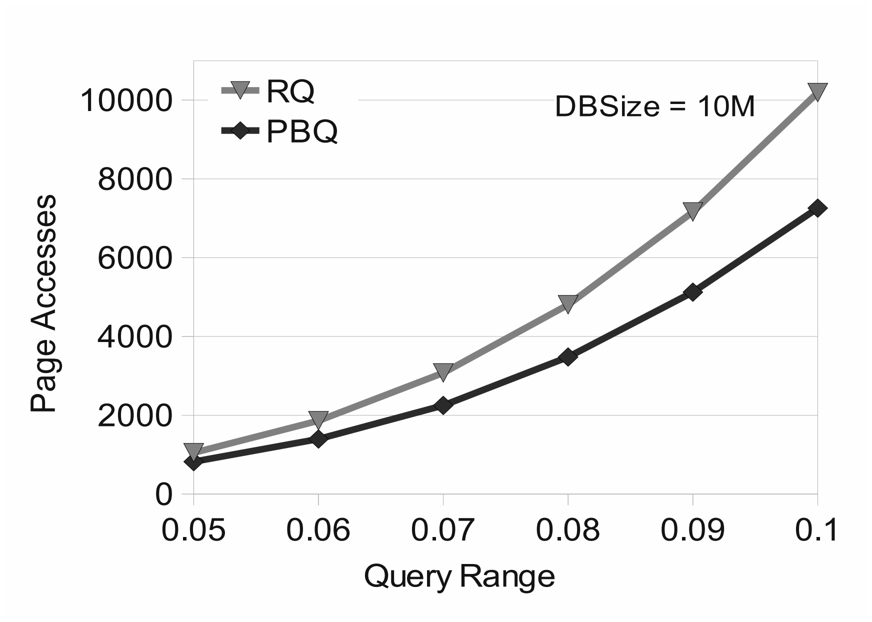


Figure 3.14: Effect of range on page accesses

records (obtained from a GIS company). We randomly selected 100 points from the database as range query centers. For each query center, range was changed from 0.01 to 0.05. Figure 3.15 shows that even for this small database pruning box query has better performance than the traditional range query on the packed R-Tree.

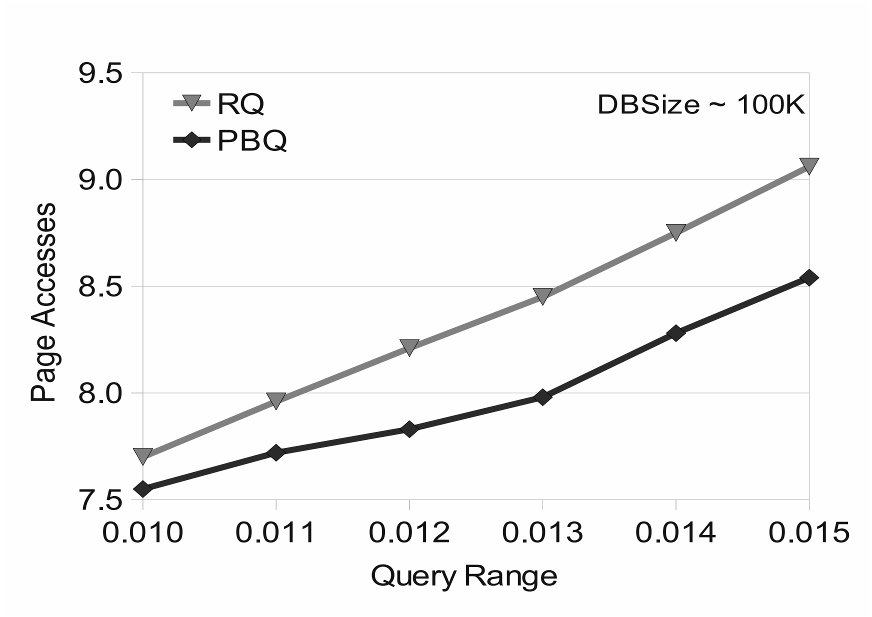


Figure 3.15: Range query on GIS data

Our second dataset is an image feature dataset. Similar to the features used in Coral Image data[59], first three moments were calculated for hue, saturation and intensity values of the pixels. Each image is described using this 9-dimensional feature vector. A feature database of 10 million images obtained from [49] was built. Range was varied from 0.005 to 0.010. Note that the value of the range was chosen such that we get a reasonable number of hits. Figure 3.16 shows that with increasing range, performance improvement due to PBQ increases. which highlights applicability of the proposed transformation to high-dimensional spaces.

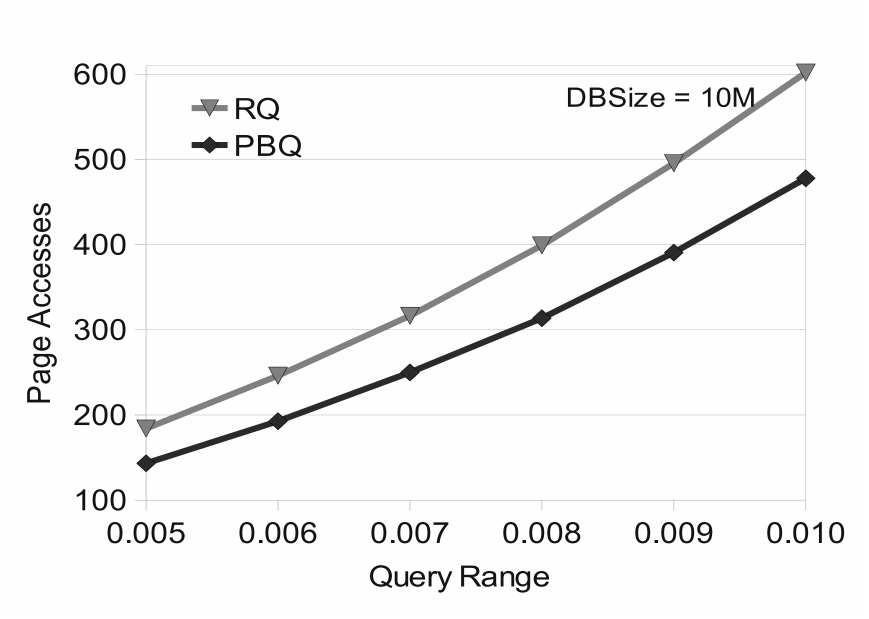


Figure 3.16: Range query on 9-dimensional image feature data

3.8 k-NN queries using transformation

k-NN (k-Nearest neighbor) queries can be considered as a special case of range queries which only return the set of k records that are most similar to the query record. A k-NN query can be implemented by keeping track of distance d_k of current k^{th} neighbor [34, 57]. Any data node whose MBR is farther than d_k can be safely pruned. This is conceptually similar to range query with range d_k. As query execution proceeds, the query range decreases (i.e. query sphere shrinks). Due to the underlying similarity between k-NN queries and range queries the proposed transformation can also be applied for k-NN queries. The algorithm used for implementation of k-NN query in the original space is described in section 2.3.1.3. The same algorithm can be adapted for use in transformed space, by creating a box query whose side shrinks as the search proceeds.

It was observed that at higher levels in the index tree, node MBRs are large in size and query point is inside a lot of MBRs and its minimum distance from these MBRs is zero. We break the ties in such cases using distance of the query point from the center of the MBR. This heuristic was observed to perform better than choosing any one of the qualifying MBRs randomly.

In the next few subsections we present results of using the transformation for k-NN queries. We executed k-NN queries in the transformed space as well as in the original space. Different combinations of number of dimensions, database size and value of k were used. In all th following graphs, the labels KNN-Orig and KNN-Trnf are used for representing k-NN queries in original space and transformed space respectively.

3.8.1 k-NN in 2-D Space

Figure 3.17 shows the results of running 500-NN queries on 2-D database. Database size was varied from 2 million to 10 million records. It can be seen that k-NN query in transformed space is consistently better than that in the original space. The slight jump in the number of query I/O for the database size of 8 million is due to the increase in the height of the tree.

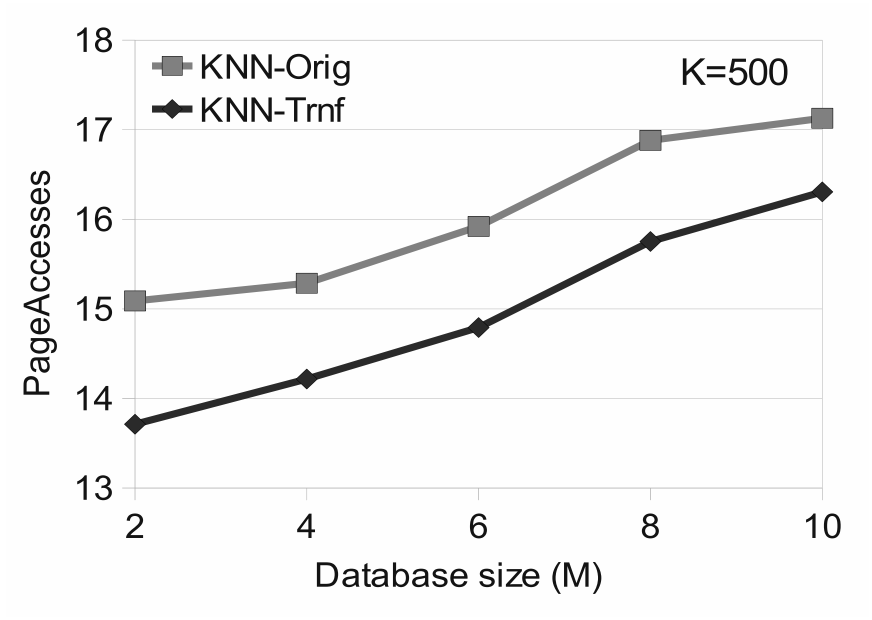


Figure 3.17: k-NN queries in 2-D for various database sizes

Figure 3.18 shows the results of varying the value of k (i.e. number of neighbors) for a database of 10 million records. We observed that for very small values of $k(< 100)$, queries in transformed space perform worse than those in original space. This is expected because for small k the effective radius of the query space is much smaller. From equations 3.6 and 3.8, it can be seen that for small query radius, the transformation may not be very effective. So our observations are in accordance with the analysis. The performance improvement was observed to be around 7% for larger values of k.

3.8.2 k-NN in 10-D Space

Figures 3.19 and 3.20 highlight advantages of transformation for K-NN queries in 10-dimensional space. It can be clearly seen from the figures that improvement in excess of 20% can be achieved using the transformation. Further, the improvement increases with increasing

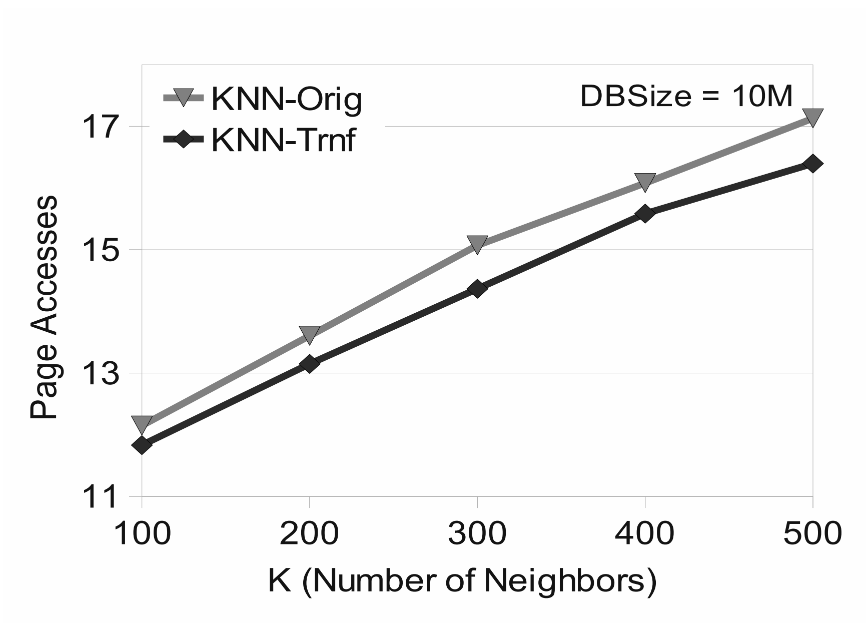


Figure 3.18: k-NN queries in 2-D for various values of K

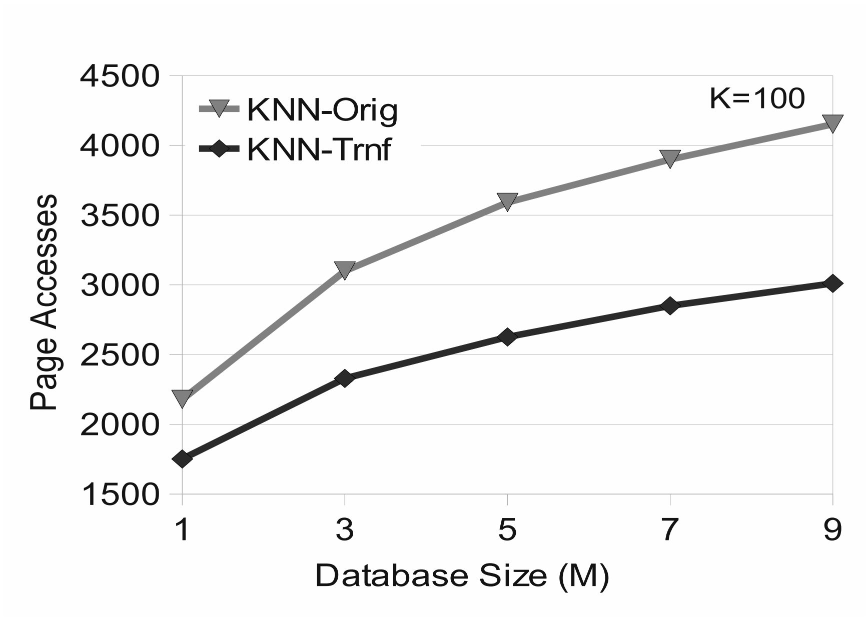


Figure 3.19: k-NN queries in 10-D for various database sizes

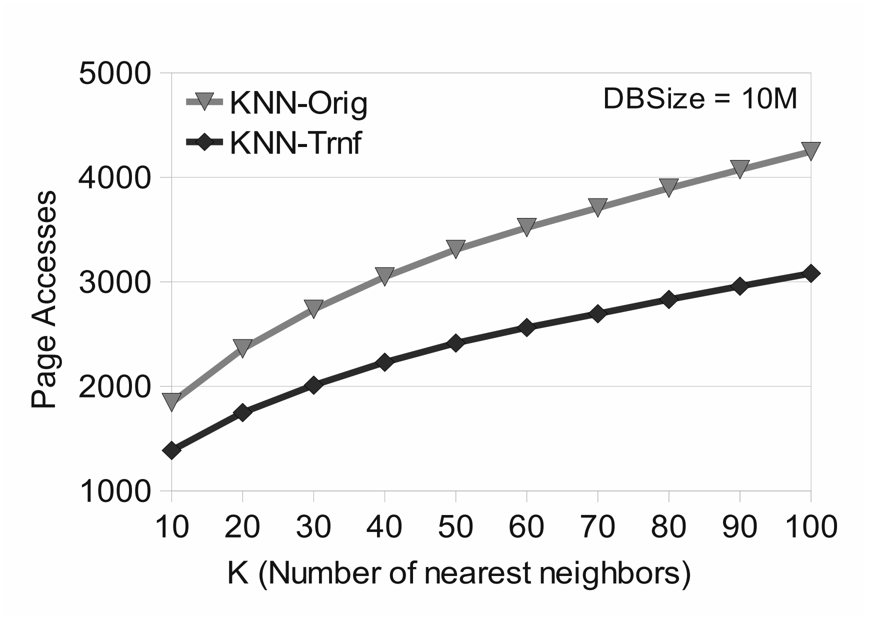


Figure 3.20: k-NN queries in 10-D for various values of K

database size and increasing values of k.

3.8.3 k-NN on Real Data

We used the proposed technique for the real databases mentioned in section 3.7. Our findings are in accordance with the observations we made with synthetic data. For 2 dimensional real data, there is no improvement if the value of k is small. But for $k \geq 80$, the transformation indeed provides better performance. For the 9-dimensional image feature data, as the value of k increases, query I/O for both the methods increases. However, rate of increase of query I/O is much slower in transformed space. It can be seen that we get about 25% improvement in the query I/O which highlights effectiveness of the transformation.

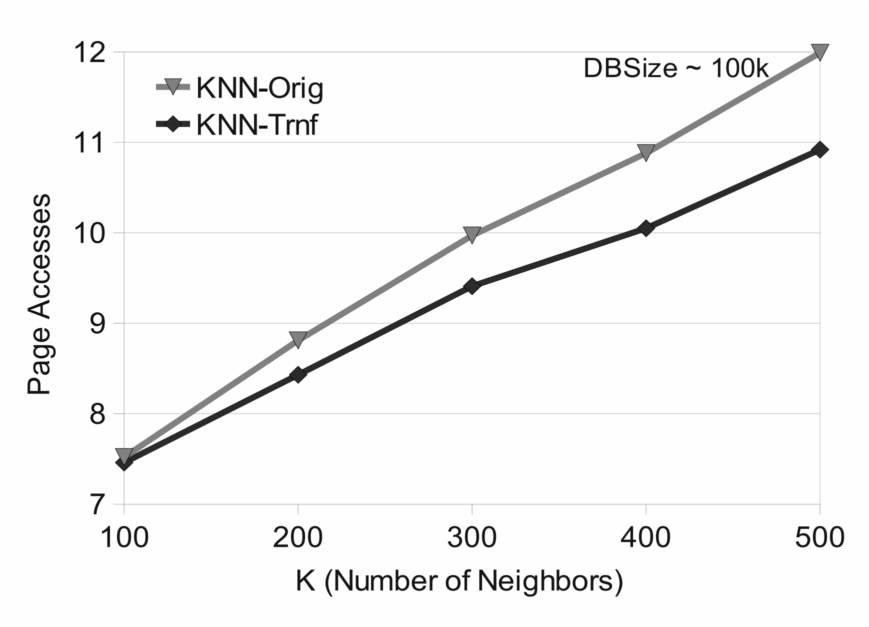


Figure 3.21: k-NN queries on GIS data for various values of K

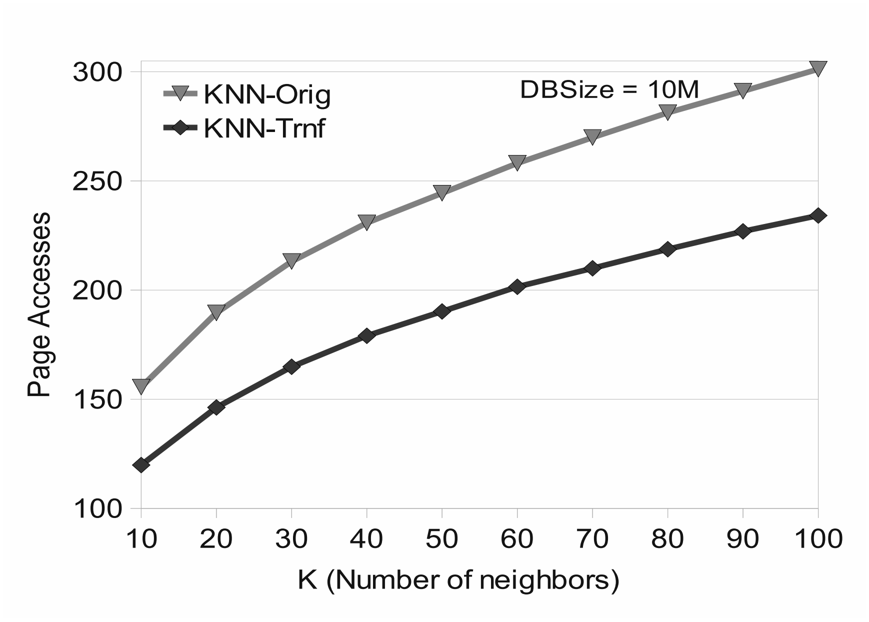


Figure 3.22: k-NN queries on image feature data for various values of K

Chapter 4

Transformation From Box Query To Range Query

The previous chapter discusses mapping of range queries and k-NN queries to box queries for efficient implementation in R-tree based structures. However, there has been a lot of research in distance based indexing schemes that use the distances among the data points for building the index. M-tree[19] described in section 2.3.3 is one such indexing scheme. Being a distance based index, M-tree provides efficient implementation of range queries and k-NN queries, however, it is not very effective for box queries. In this chapter we will discuss mapping of box queries to L_1 range queries for their efficient implementation in M-tree like indexing schemes.

4.1 Data and query transformations

The basic idea behind the transformation is similar to the axis aligning rotations discussed in the previous chapter. However, length of sides in a box query can be non-uniform which

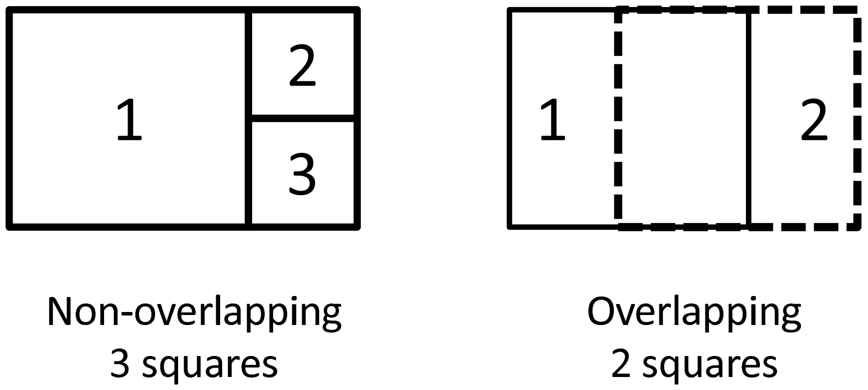


Figure 4.1: Square tiling of a rectangle

poses challenges for direct mapping of box queries onto range queries.

4.1.1 2-Dimensional Transformation

A two dimensional box query is a rectangle while a range query (in L_1) is a diamond. If the box query is a square, a simple rotation by 45 degrees transforms a box query in to a range query. However, when the sides of the box are not uniform, the transformation is slightly involved. If the rectangle can be expressed as a combination of squares (the process is called square-tiling), then each component square can be transformed into a range query. As shown in figure 4.1 there are a number of ways for square tiling of a rectangle. When the squares that make up the given rectangular box query overlap each other, we call it an *overlapping transformation*; and similarly the transformation that converts a rectangle into a set of non-overlapping squares is called a *non-overlapping transformation.*

For a non-overlapping transformation, we can find a minimal square tiling of the given rectangle using the Euclidean tiling algorithm and a recursive construction technique in [42]. However, the minimum number of squares required for a non-overlapping transformation may be fairly high. Each square component of the box query is mapped to a range query whose execution needs resources such as CPU and memory. Hence, in order to minimize the resource requirement, we have to minimize the number of range queries. Hence, for both overlapping and non-overlapping transformations, we want to minimize the number of squares that make up the given rectangle. Note that neither of the transformations have any false positives. Hence, in a disk based database, non-overlapping transformation does not provide any advantage over overlapping transformation. However, overlapping transformation generally results in smaller (and deterministic) number of squares. Hence, we restrict ourselves to overlapping transformation in this dissertation. For an overlapping transformation, we can convert a rectangle of size $l \times b \;\; (b < l)$ with a minimum of $\lceil \frac{l}{b} \rceil$ overlapping squares of side length b. The basic idea is to tile the rectangle with $b \times b$ sized non-overlapping squares until we are left with a rectangle of size $(l \; mod \; b) \times b$. This region is then covered by a $b \times b$ square that overlaps with the previous square (see figure 4.1 for an example). The theorem below proves that the overlapping transformation provides tiling with minimum number of squares.

Theorem 4.1.1. *Overlapping transformation gives minimum square tiling of the box query.*

Proof 4.1.1. *The largest square that can be used to tile a rectangle of size, $l \times b \;\; (b < l)$ has size $b \times b$. Hence the minimum number of squares needed $= \left\lceil \frac{\text{area of the rectangle}}{\text{area of the square}} \right\rceil = \lceil \frac{lb}{b^2} \rceil = \lceil \frac{l}{b} \rceil$ = number of squares used by our transformation.*

Algorithm 1 presents an extension of existing box query algorithm to execute the multiple box queries in existing index structures based on bounding boxes (such as R*-Tree [7]) or bounding spheres (such as M-tree [19]). The main idea is to compare each MBR (or hyper-sphere in case of M-tree) with all the box queries. If the MBR overlaps with any of the queries, then its corresponding node needs to be searched. If no box query overlaps with the

MBR then its subtree can be safely pruned.

Algorithm 1 To implement a box query B using a set of box queries $B_1, B_2 \dots B_n$.

Input : A Box query B which has been tiled into multiple queries B_1, B_2, ... B_n each of which is a square, and a node N in the database index tree.
Output : A list of database objects satisfying the box query B.
Algorithm :

```
 1: if $N$ is a leaf node then
 2:    for each object $o$ in $N$ do
 3:       if $o$ is inside at least one of the queries $B_1 ... B_n$ then
 4:          Add $o$ in the result set
 5:       end if
 6:    end for
 7: else
 8:    for each entry $e$ in $N$ do
 9:       if $e$ overlaps with at least one of the queries $B_1 ... B_n$ then
10:          Search $e$ recursively.
11:       end if
12:    end for
13: end if
```

An important property of this algorithm is that the query I/O for the multiple square box queries is exactly the same as that required for the original box query.

Theorem 4.1.2. *Algorithm 1 has the same I/O cost as that of the original box query.*

Proof 4.1.2. *The only time algorithm 1 accesses an index node N is when the index page overlaps with one of the sub-queries (say B_i). The sub queries do not have false positives or false negatives (i.e. they do not have any extra query space neither do they miss any part of the original query). Hence, the fact that N overlaps with B_i implies that N overlaps with B. Similarly, if N overlaps with B then it must overlap with B_j for some $1 \leq j \leq n$. Note that N is accessed at most once even though there may be multiple B_j overlapping with N. Hence, each node N accessed in the given algorithm will also be accessed in the traditional box query and vice versa. Thus, the two queries have the same I/O cost.*

Once a box query is mapped to multiple square box queries, each of the square box queries can be transformed into corresponding range query. These range queries can then be run using algorithm similar to algorithm 1. As this algorithm keeps multiple queries in main memory it will have slightly higher main memory cost and CPU cost. But the similarity in query space and index page space provides improvement in the number of page accesses required for query execution.

4.1.2 High Dimensional Transformation

The space rotation approach discussed so far does not work directly for data with more than 2 dimensions. We instead use the concept of Disjoint Planar Rotations from the previous chapter. Each 2-D plane of the input data space is transformed using rotation. Each box query is also projected on 2-D planes to generate a combination of box queries. Let B be a box query in n dimensions and let $B_{i,j}$ be its projection on i-j plane. A point $p = (p_1, p_2 ... p_n)$ satisfies the box query if its projections satisfy the projected boxes on each of the 2-D planes. Formally,

$$p \in B \Longleftrightarrow p_{1,2} \in B_{1,2} \cap p_{3,4} \in B_{3,4} \ldots \cap p_{n-1,n} \in B_{n-1,n}$$

Here, $p_{i,j}$ denotes project of p in $i - j$ plane.

Each projection of B is a 2-D box query which can be executed using method outlined in the previous section. Recollect that M-tree stores one radius for each directory entry. This radius is the radius of the data sphere represented by the page and it is calculate with respect to all n dimensions. As our individual queries are in 2-D space, use of global radius may not provide much pruning and thus makes the query execution extremely inefficient. We instead propose a minor modification in the structure of M-tree that can overcome this inefficiency. For each directory entry in the M-tree we also keep track of radii of the projections of the data sphere. On a flip side this approach slightly reduces fan out of the M-tree directory node (due to the space required to keep this extra information), but, it provides much better pruning power, which results in lower number of page accesses.

4.2 Improvement Estimation

We now present analysis of 2-dimensional transformation that can be used to estimate performance improvement due to transformation. We assume that the data is normalized in unit cube and is indexed using M-tree using L_1 distance measure. Hence, each index page geometrically is a diamond. Further, we also assume that all the index pages have the same radius and are uniformly distributed in the data space (i.e. probability of having an index page centered at a random point in the space is the same irrespective of location of the point). We compute area of the region in which the center of the index page must lie in order to overlap with the query space. For uniformly distributed data this area can be used to estimate the probability of overlap between an index page and a query. We begin with analysis of 2-D queries.

4.2.1 Analysis of 2-D Queries

With reference to figure 4.2, let input box query has size $l \times b$. The shaded region in the figure highlights the space in which center of any index page of radius r must lie in order to overlap with the box query. Hence the probability P_{bq} that a randomly selected index page

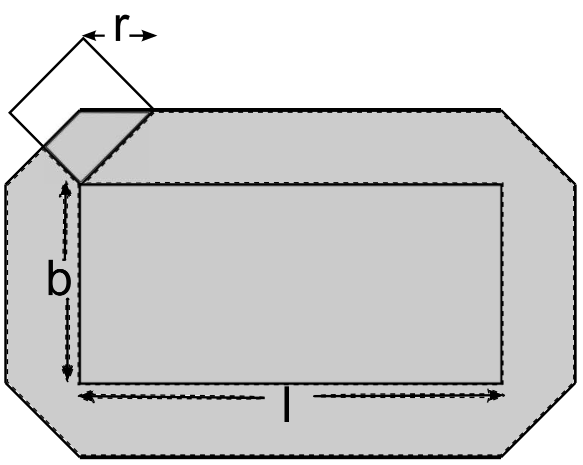


Figure 4.2: Box query analysis

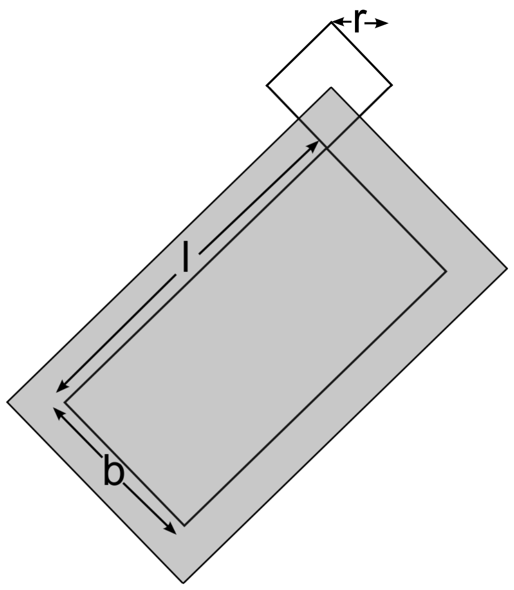


Figure 4.3: Transformed range query analysis

will overlap with box query is,

$$P_{bq} = \frac{\text{Area of the shaded region}}{\text{Area of the data space}} \tag{4.1}$$

$$= \frac{lb + 2r(l + b) + 2r^2}{1} \tag{4.2}$$

Now consider figure 4.3 for corresponding range query. Note that as the overlapping transformation does not have false positives hence the query space remains the same (rectangle of size $l \times b$). For the same size of the index pages, the center of the index page must lie in the shaded region in the figure in order to overlap with the query space. Hence the probability P_{rq} that a randomly chosen index page overlaps with the query space is,

$$\begin{aligned} P_{rq} &= \frac{\text{Area of the shaded region}}{\text{Area of the data space}} \qquad (4.3) \\ &= \frac{lb + r\sqrt{2}(l+b) + 2r^2}{1} \qquad (4.4) \end{aligned}$$

It can be seen from equations 4.2 and 4.4 that for the given values of l, b and r, the range query will have slightly smaller probability of overlap. Hence, the transformation is expected to have lesser number of page accesses resulting in better query performance.

4.2.2 Analysis of High Dimensional Queries

An important characteristic of box queries is that overlapping of query space and index pages is dimension independent. What we mean by this is, whether a query overlaps with an index node along dimension i does not depend on their overlap along any other dimension $j (j \neq i)$. Hence, probability P_{bq} of overlap between a n dimensional box query is calculated as,

$$\begin{aligned} P_{bq} &= P(\text{Query overlaps with the index page along all the dimensions}) \\ &= P(\bigcap_{i=1}^{n} \text{query overlaps along dimension } i) \\ &= P(\bigcap_{i=1}^{n/2} \text{query overlaps in the projection in each 2-D plane}) \qquad (4.5) \end{aligned}$$

Due to dimension independence property, this can be simplified to,

$$P_{bq} = \prod_{i=1}^{n/2} P_{bq}^{(2i-1),2i} \tag{4.6}$$

where, $P_{bq}^{2i-1,2i}$ is probability of overlap along dimensions $2i-1, 2i$ obtained using equation 4.2.

The concept of dimension independence also applies to the transformed range queries as the range queries in different 2-dimensional projected planes are not related. Hence, expression for probability that a transformed range query overlaps with the index page is,

$$P_{rq} = \prod_{i=1}^{n/2} P_{rq}^{2i-1,2i} \tag{4.7}$$

where, $P_{rq}^{2i-1,2i}$ is obtained using equation 4.4. Since each term in equation 4.7 is smaller than corresponding term in equation 4.6, we can expect P_{rq} to be much smaller than P_{bq}.

4.3 Experimental Results

In this section, we present the results of applying the proposed transformation to synthetic and real data. All the experiments are run on AMD Opteron 2.2 GHz systems running GNU/Linux. Synthetic data containing a mixture of Gaussian clusters (with random means and covariance matrices) are generated in 2 and 10 dimensional space. M-tree is built in both original space and transformed space (with L_1 distance metric). Box query (in original space) and its equivalent range query (in transformed space) is run to measure the number of disk page accesses.

We also compare performance statistics of M-tree with linear scan (i.e. data stored in flat files without any indexing) wherever they are comparable. As suggested in [65], due to sequential disk access, linear scan is 10 times faster than random disk access. Hence, we used 10% linear scan (i.e, 0.1 $\times$ Number of disk pages in a flat file) in our comparison. However, in cases where linear scan is considerably worse than indexing, it is dropped from the graphs for better clarity.

In all the figures we used the label BQ to denote M-tree running traditional box query, MRQ to denote M-tree running multiple range queries and 10%L to denote linear scanning.

4.3.1 2-D Transformation

In the first set of experiments we consider 2-dimensional synthetic data. We present results with uniform query box size (all the sides of the box are of the same length) as well as with random query box size (sides of the box are chosen randomly). Note that, a uniform box query is a (hyper)square and can be mapped to single range query. The concept of multiple queries comes into play only when the box query is non-uniform.

Figure 4.4 shows the effect of database size. As per our analysis in section 4.2.1, although we see improvement in the query performance, it is relatively low (query I/O reduction is about 8% to 12%). But it should be noted that as the database size increases, transformation tends to perform better and better which is very encouraging.

Figure 4.5 shows performance improvement with increasing query box size. As the box size increases, larger and larger volume of the database is queried, hence I/O for both the methods increases. However, it can be seen from the figure that the transformation is more and more effective with increasing query box sizes (5% I/O reduction for box size 0.01 to 9% for box size 0.05).

In figure 4.6, we demonstrate effect of increasing database size, on box queries with

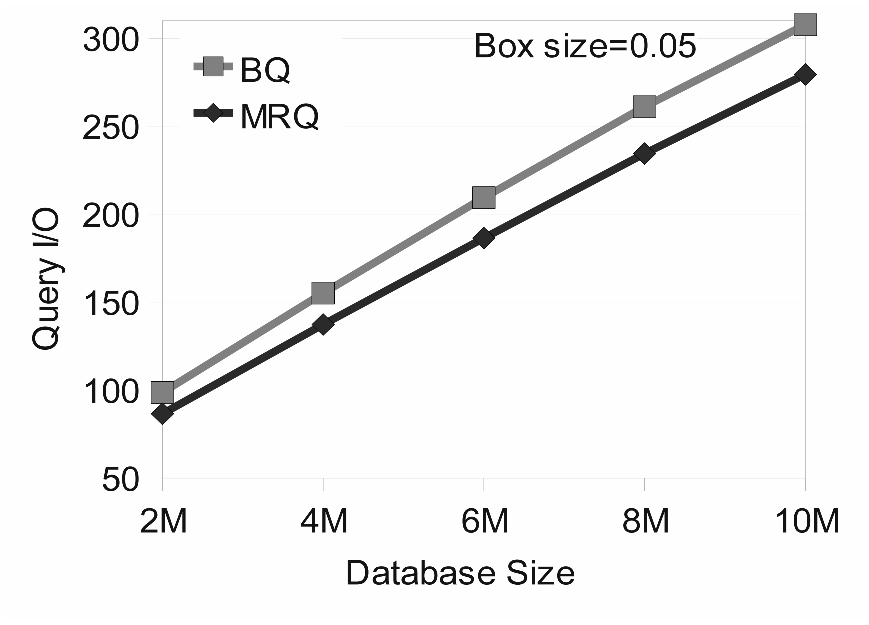


Figure 4.4: 2-dimensional data with varying database size

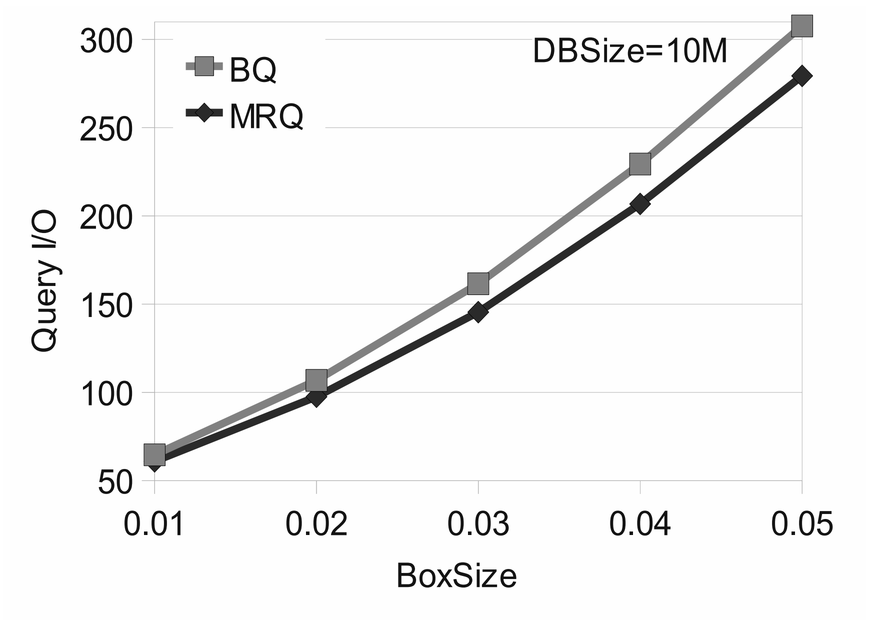


Figure 4.5: 2-dimensional data with varying box size

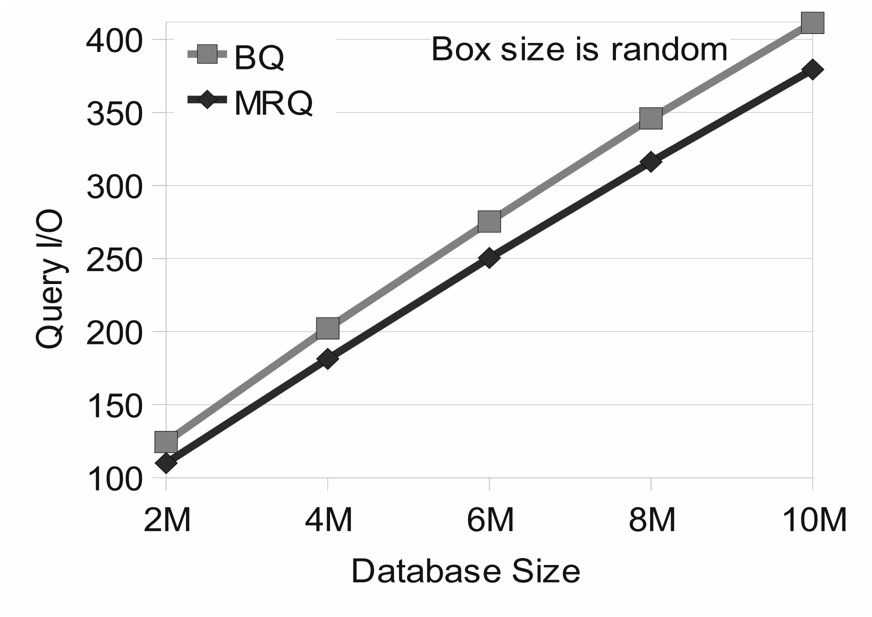


Figure 4.6: 2-dimensional data with random box size

random box sizes. In each of the experiments, box sizes were chosen randomly. However, maximum box size was limited to limit the number of hits. It can be seen that even in this case, as the database size increases, the transformation provides better improvement.

4.3.2 High Dimensional Transformation

We now present results of applying the proposed transformation to 10-dimensional synthetic data. Just like the previous case, we compare the performance of our transformation by varying database sizes and box sizes.

Figure 4.7 shows box query I/O with increasing database size. Box size was kept at 0.2 and the boxes were all uniform. It can be seen from the figure that effectiveness of transformation increases with increasing database size. For, database with 10 million records, it reduces query I/O by 30%.

Figure 4.8 shows effect of increasing box size. For very small box sizes, transformed

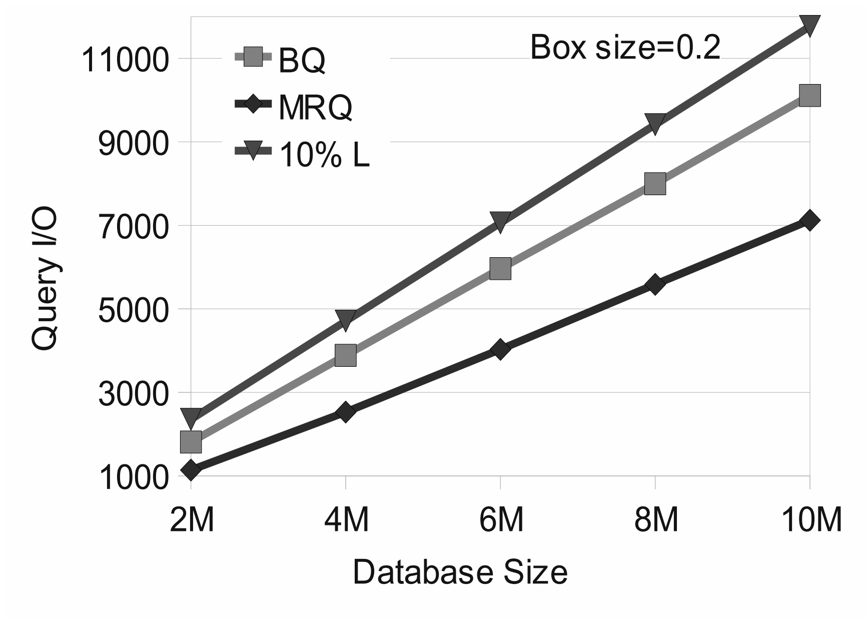


Figure 4.7: 10-dimensional data with varying database size

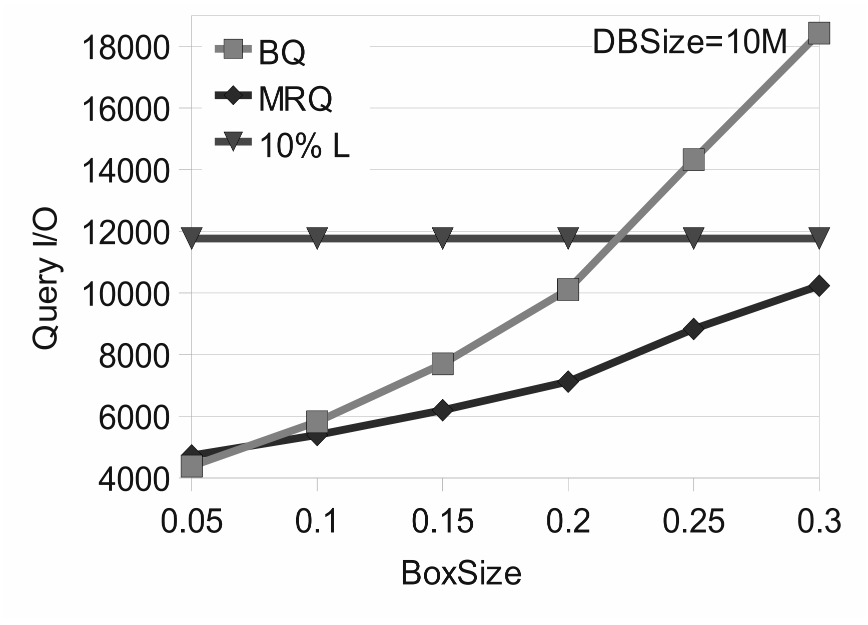


Figure 4.8: 10-dimensional data with varying box size

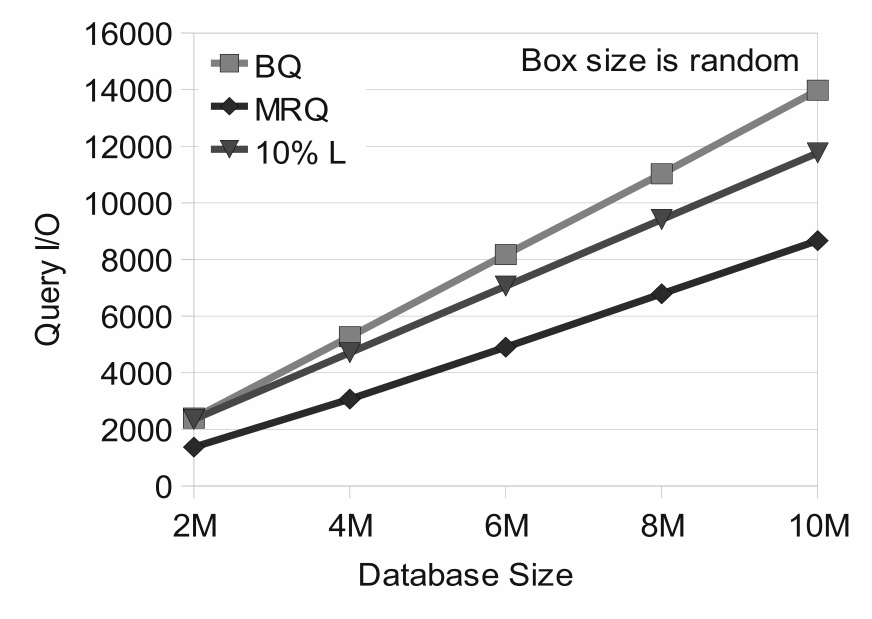


Figure 4.9: 10-dimensional data with random box size

M-tree is slightly worse than the traditional M-tree. But as box size increases, performance gain due to to transformation increases. At very large box sizes, 10% linear scan starts outperforming the indexing schemes. This is expected as the large boxes tend to retrieve much larger percentage of data. It is still encouraging to see that for box size 0.3, even when traditional M-tree performs considerably worse than linear scan, the transformed M-tree maintains its better performance.

Figure 4.9 shows effect of running random sized box queries with increasing database sizes. It can be seen that, traditional M-tree cannot compete with linear scan for small database sizes, however, transformed M-tree consistently outperforms other schemes.

4.3.3 CPU Consumption

As the transformed M-tree executes multiple range queries for the same box query, it is expected to take more CPU time than the original M-tree. Table 4.1 compares the CPU

Table 4.1: Comparison of CPU times

# Dimensions	**CPU times (milli-seconds)**	
	BQ	**MRQ**
2	2.1	54.7
10	45.8	464.1

times for original and transformed queries for 2 and 10 dimensional databases containing 10 million records. For uniform box queries, there is exactly one range query for each box query (ergo, CPU cost is similar). Hence, random box sizes were used for these experiments. As expected the transformed M-tree takes much more (10X - 27X) CPU time than the traditional one. However, in a very large database, the time saved due to reduced I/O can easily make up for extra time lost in CPU computation. An interesting observation to note is that CPU time comparisons are much better in 10-D data than in 2-D data. This may appear counter intuitive at the first glance, but it should be remembered that in 10-dimensional space, the individual radii maintained in M-tree provide a much better scope for pruning which avoids several unnecessary box comparisons resulting in (relatively) better CPU consumption.

4.3.4 Experiments with Real Data

In this section, we present results of applying the proposed transformation on image feature database. The database is essentially the same database used in sections 3.7 and 3.8.3. It contains 50 million images, each with 9 features (first three moments for hue, saturation and intensity).

Figure 4.10 shows box query I/O with increasing database size. The box size was kept at 0.2 and the boxes were all uniform. It can be seen that the transformation provides about 14% I/O reduction for the database with 10 million feature vectors. Further, the performance

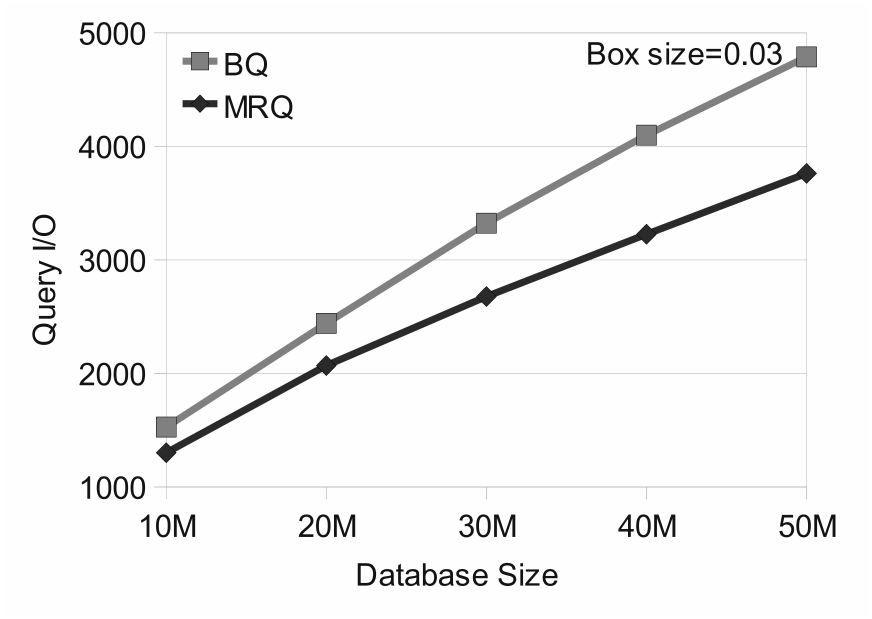


Figure 4.10: Image database with varying database size

improvement increases with increasing database size (about 21% for 50 million).

Figure 4.11 shows effect of increasing box size for the database containing 50 million feature vectors. Similar to our observations with synthetic data, for very small boxes the transformed M-tree performs slightly worse than the traditional M-tree. However, as box size increases, performance gain due to to transformation increases. For the largest box size the I/O reduction is about 28%.

Figure 4.12 shows effect of running random box queries with increasing database sizes. The transformed M-tree comprehensively beats the traditional M-tree providing I/O reduction of about 80% which increases with increasing database size. Queries with random box sizes are more realistic than those with uniform box sizes in practical application. The excellent performance gain for random box queries highlights importance of transformation in high dimensional data spaces.

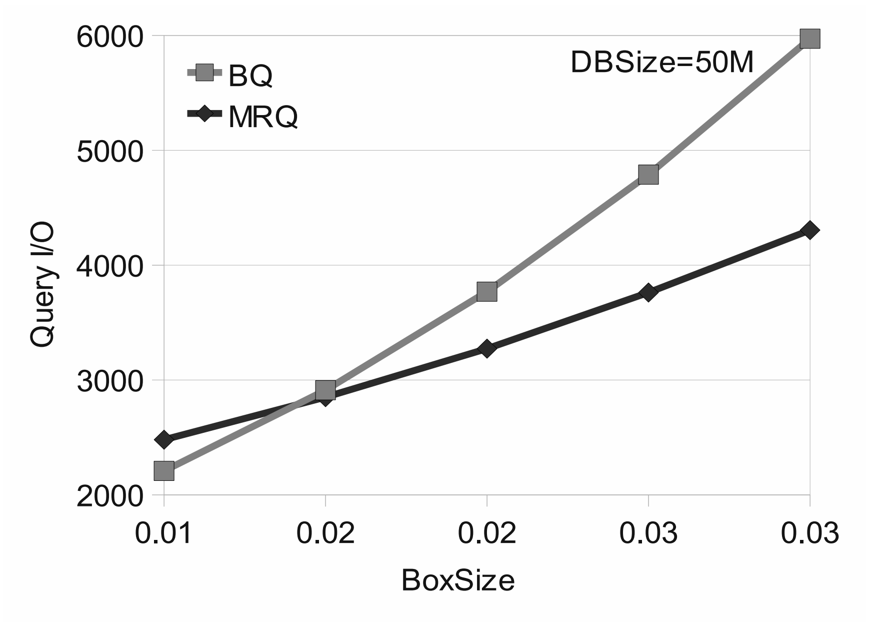


Figure 4.11: Image database (50M) with varying box size

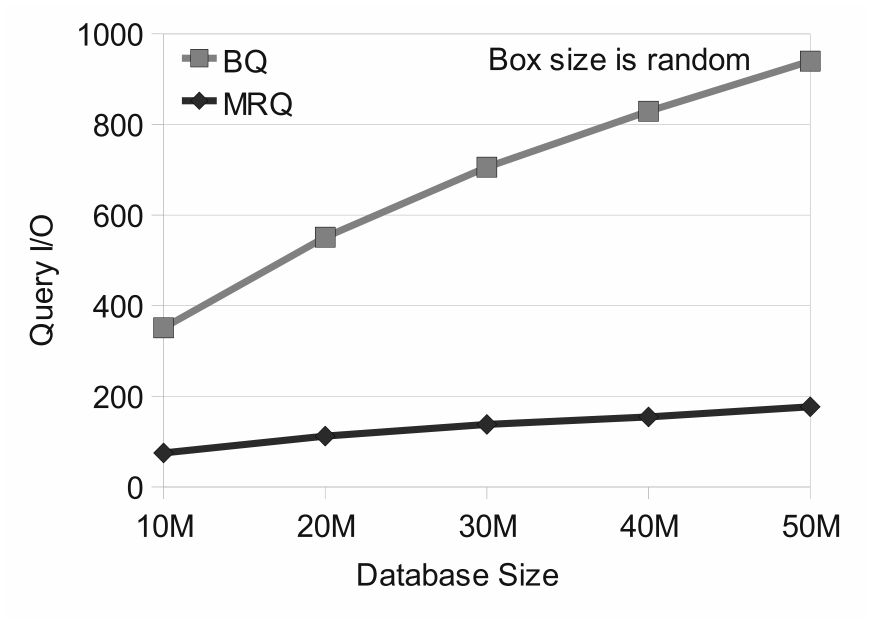


Figure 4.12: Image database with random box size

Chapter 5

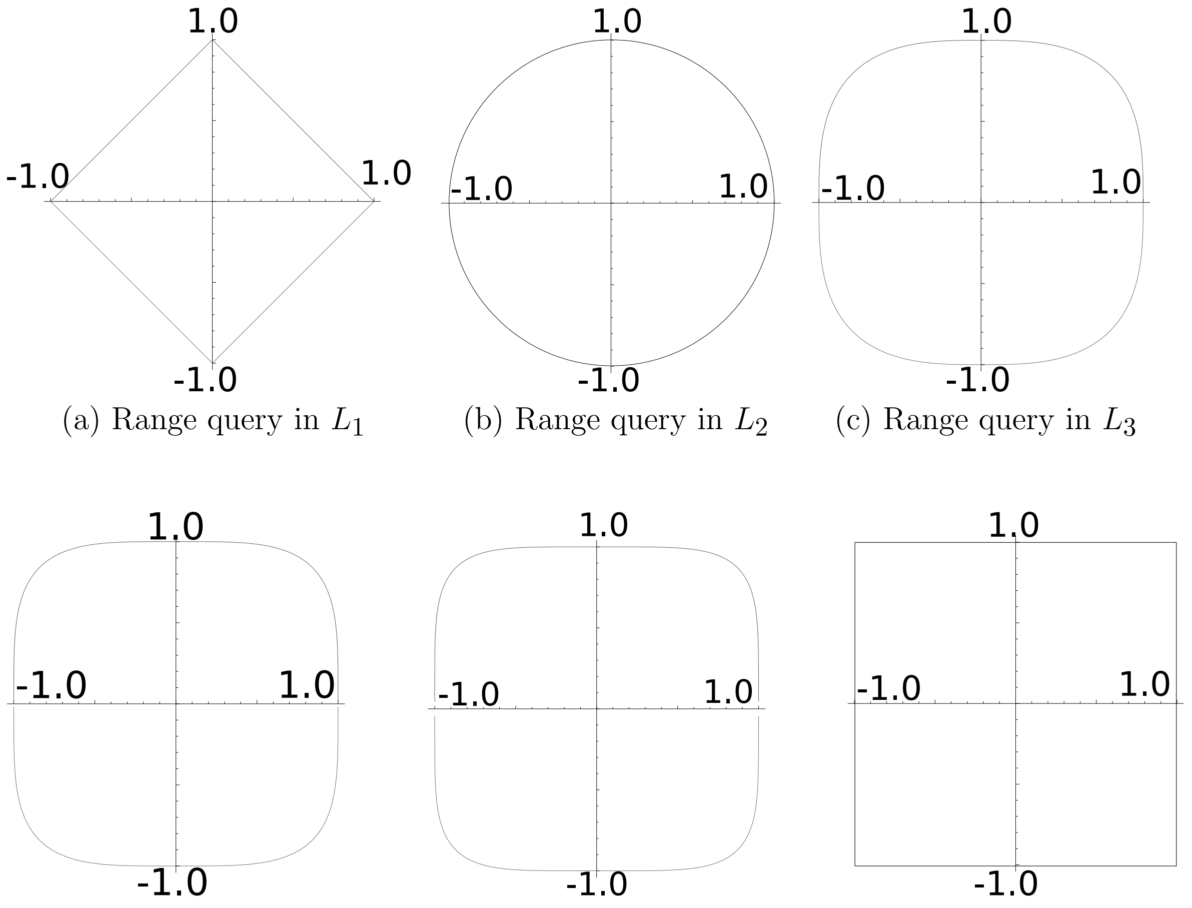
Transformation In L_2 Space

The transformations discussed so far apply to data in L_1 space. However, Euclidean distance (also known as L_2 norm) is another equally popular distance measure used in information retrieval systems. In this chapter we will consider the problem of applying transformation approaches in L_2. We begin the discussion with geometry of L_p range queries for various values of p.

5.1 Shapes for higher order L_p queries

We have already introduced generic L_p norm (also known as Minkowski norm) in equation 1.1. In this section we analyze how the shapes of the range queries change with the value of parameter p.

Figure 5.1 shows range queries in 2-dimensional L_p spaces for $p = 1, 2, 3, 4, 5, \infty$. In each of the queries, center of the query is fixed at origin and radius is fixed at unity. The first observation that can be made from the figure is that L_{p+1} query is a superset of L_p query. In other words, each higher order query contains all the points in the lower order query plus



(a) Range query in L_1 (b) Range query in L_2 (c) Range query in L_3

(d) Range query in L_4 (e) Range query in L_5 (f) Range query in L_∞

Figure 5.1: Variation in the query space with increasing Minkowski parameter p

a few additional points. The second observation is that the extra points get added more along the 45°lines. In fact, the point along the axes are the same for all the queries. Third and the most important observation is that with increasing order the queries resemble more and more to a square box query i.e., the query space becomes more and more squarish. As a matter of fact, the query space corresponding to the L_∞ space is exactly a square. Thus, as the order p of the Minkowski norm increases, the range query becomes more and more box like.

5.2 Mapping L_2 range query to box query

As mentioned earlier, for $p = \infty$, range query is in fact the same as the box query. Thus, in order to map a range query in L_2 to a box query in L_∞ we have to come up with a mapping that converts L_2 sphere to L_∞ sphere.

A theorem by Gromov [28] states that two compact metric spaces which have Gromov-Hausdorff distance[29] zero from each other are isometric i.e., we can map spheres from one space to those in the other such that there are neither false positives nor false negatives. However, the curvature of Euclidean space is 0. On the other hand the curvature of the L_∞ space is negative. Hence, the Gromov-Hausdorff distance between them is non-zero meaning they are not isometric. This essentially means that it is impossible to precisely map spheres (or range queries) in L_2 space onto spheres in L_∞ space. As L_∞ range queries are geometrically similar to box queries, we conjecture that transformation from L_2 range queries to box queries may not be possible.

5.3 Approximate transformations

Since the exact transformation from L_2 range query to a box query may not be possible, we consider approximate transformations. Desired properties of an approximate transformation are as follows :

- The transformation should miss very few (ideally zero) true positives.
- It have very few false positives.
- The transformation should map the circular range queries to the rectangular box queries.

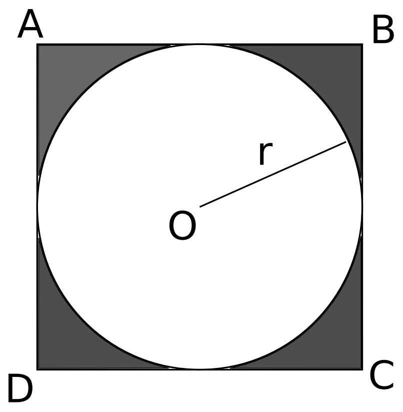


Figure 5.2: False positives in a minimum bounding box query in 2-D

In the next few subsections, we discuss some of the possible approximate transformations. We begin with a simple minimum bounding box query and then move on to more involved transformations. We will assume 2-D space unless explicitly specified otherwise.

5.3.1 Minimum bounding box query

The simplest form of mapping between a range query and a box query is to approximate range query using a bounding box. Although the bounding box is guaranteed to return all the true positives, it also returns a lot of false positives. For uniformly distributed data, the amount of false positives is proportional the area of the extra space queried by the bounding box query.

Figure 5.2 shows the L_2 range query at point O with radius r, which is approximated by a box query $\square ABCD$. The shaded region in figure 5.2 corresponds to the false positives. The amount of false positives can be estimated as,

$$
\begin{aligned}
FP &= \frac{Area\ \ of\ \ shaded\ \ region}{Area\ \ of\ \ query\ \ space} \\
&= \frac{Area\ \ of\ \ \Box ABCD - Area\ \ of\ \ query\ \ space}{Area\ \ of\ \ query\ \ space} \\
&= \frac{4r^2 - \pi r^2}{\pi r^2} \\
&= \frac{4-\pi}{\pi} \\
&\approx 27.32\%
\end{aligned}
$$

As the number of dimensions increases, the percentage false positives increases exponentially. Note that each false positive can potentially result in an unwarranted disk page access. Hence, use of minimum bounding box to approximate a range query becomes inefficient with increasing dimensions.

5.3.2 Elliptical range queries instead of circular range queries

When comparing L_2 range query with a box query, it can be observed that the curvature of boundary of the query sphere is a non-zero constant in case of range query, while it is zero in case of a box query. Hence, it may be possible to map a range query onto a box query by transforming the range query curve (i.e. a circle) to a curve with curvature close to zero. consider a simple transformation, $y' = c_y y$ and $x = c_x x$, where $0 < c_y < 1$ and $c_x > 1$. This transformation essentially shrinks one dimension while stretching the other. An L_2 circle is then mapped to an ellipse. At first glance, it may appear that since the absolute area are corresponding to the false positives is reduced, this transformation may have smaller false positives. However, it should be noted that this transformation changes the distribution of the points in the space. Since the index built will follow the distribution of the points, the index pages will now have rectangular shapes instead of squares. In other words, instead of absolute area, we should consider area of false positives relative to the area of the range query as a true measure of index performance degradation. It can be easily shown that relative area corresponding to the false positives remains the same as in the case of circular query. Hence, in practice this kind of transformation may not be used.

5.3.3 Approximating range query using multiple boxes

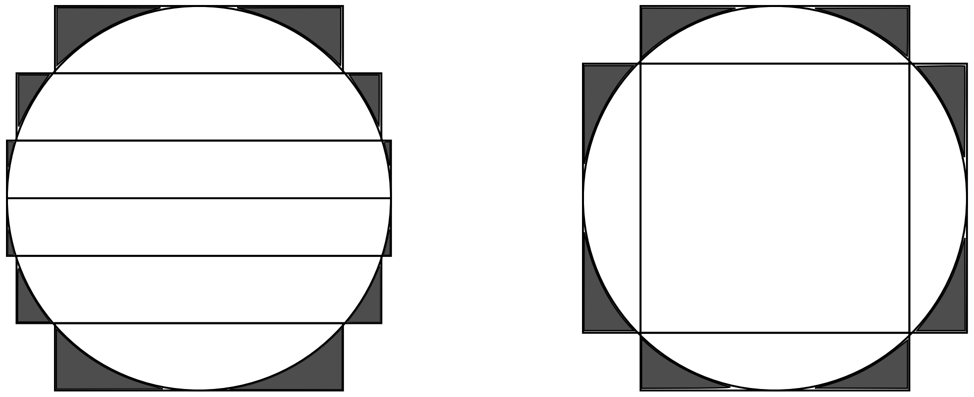


Figure 5.3: Approximating range query using multiple box queries

Another, simple extension to the simple minimum bounding box query is to use multiple boxes instead of a single box to approximate a range query. Figure 5.3 shows two possible ways in which multiple boxes can be used to approximate a 2-dimensional L_2 range query. In 2-dimensional space, using the second method of the diagram, we can reduce the percentage of false positives to 16.4%. Note that, using the first method, we can increase the number of box queries as much as we want. Figure 5.4 shows the fraction of false positives with increasing number of boxes. Here, it is assumed that all the boxes have the same width. It can be seen that with increasing number of boxes, the amount of false positives decreases.

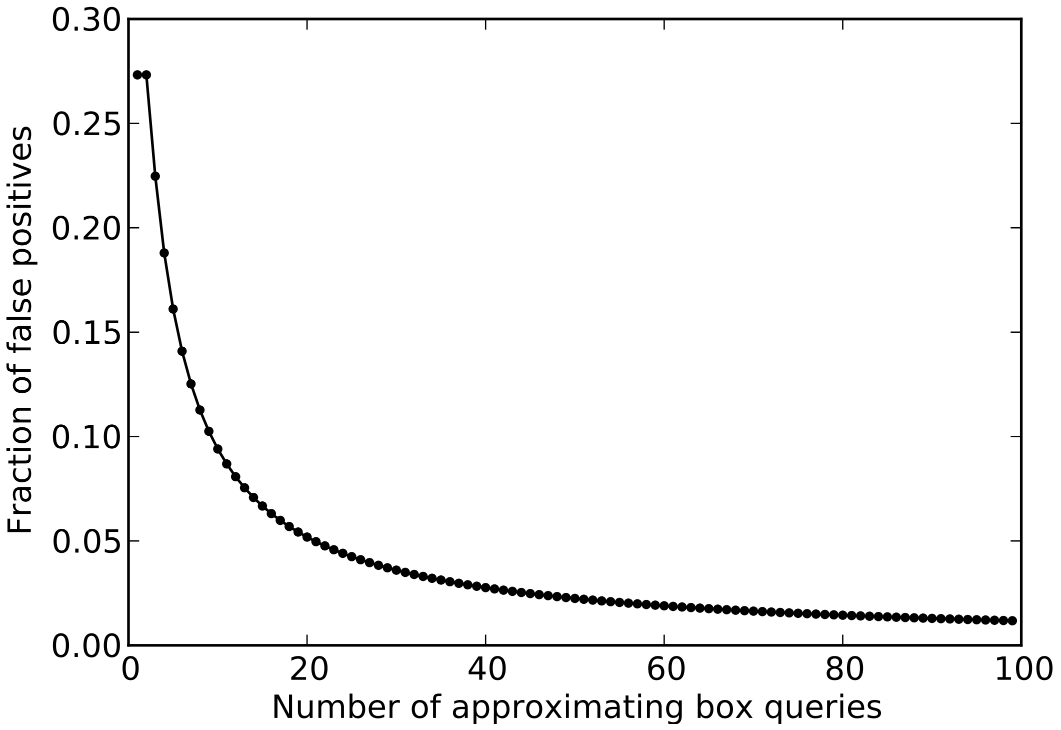


Figure 5.4: Fraction of false positives with increasing number of boxes

However, the rate of decrease in the false positives also decreases. It can be seen from the figure that even after using 20 boxes to approximate the query, we still have 5% false positives. Recollect that false positives increase exponentially with increasing number of dimensions. Further, each additional box query requires extra processing power (CPU and main memory). This may make this approach unreasonable in high dimensional spaces.

5.3.4 Using index built in L_∞ space for doing range queries in L_2 space

As mentioned earlier, range query in L_∞ is geometrically the same as a box query. Hence, it may be possible use an M-tree like index (with L_∞ norm as a distance measure) that is suitable for indexing spheres and use it to implement bounding box queries (which will not

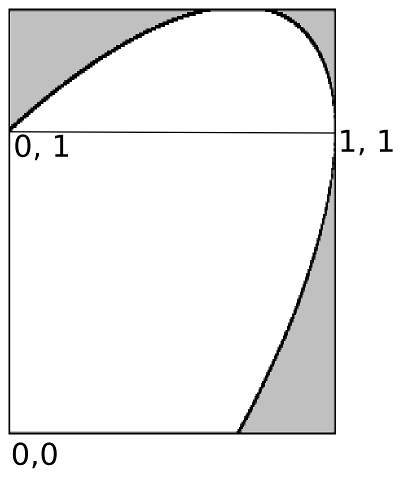


Figure 5.5: Transformation in equations 5.1 and 5.2 in first quadrant

be range queries in L_∞ norm). However, we empirically observed that M-Tree is not very effective for L_∞ distance measure and performance of M-trees running range query in L_2 space is significantly better than M-tree running range query in L_∞ space. As mentioned earlier, the number of false positives in a bounding box query increase exponentially and it is possible that even a better index structure is unable to counter the ill-effects of large amount of false positives. Further, Performance of an M-tree depends on how well it can cluster the points within a subtree. As highlighted in [3] L_2 might be a better distance measure than L_∞ to cluster points. As a result, using L_∞ range query to approximate L_2 range query may not be very effective in disk based database systems.

5.3.5 Other possible transformations in L_2 space

A good transformation from a range query to a box query will transform the curve corresponding to the range query such that it closely follows that corresponding to the box query. Motivated by this fact, it may be possible to design a transformation which follows the general form,

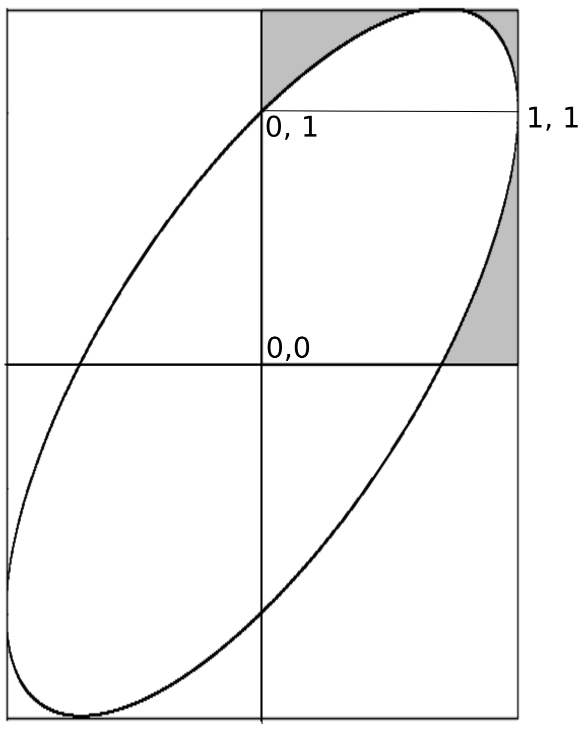


Figure 5.6: Query transformation corresponding to equations 5.1 and 5.2

$$y' = c_1 x^n + y \tag{5.1}$$

$$x' = x - c_2 y' \tag{5.2}$$

For $n = 1$, this transformation maps a circular range query to an ellipse inclined at angle with the x-axis. The shapes corresponding to higher order transformations ($n > 1$) are more complex. However, these transformations suffer from two major drawbacks

- By carefully adjusting parameters c_1 and c_2, it is possible to reduce the false positive percentage to about 5% in first and third quadrant. However, this saving comes at an expense of higher false positives in second and fourth quadrant.

- These transformations are query sensitive. In other words, not all queries have the same degree of false positive reduction and one may come up with a particularly pathological cases of queries in which the transformation may perform worse than simple bounding box queries.

Based on our extensive study of various possible transformations for L_p queries, it appears that for disk based index application it may not be possible to come up with any exact or approximate approach that can provide better performance than the original space range query. However, in main memory database applications the focus is on reduction in CPU utilization. There has been a lot of promising work on using vantage point based transformations that map range queries to box queries and is the subject of the next chapter.

Chapter 6

Space transformation using vantage points

Over the last decade or so, there has been a lot of advancement in main memory technologies and computing systems with several gigabytes of main memory are quite common. This makes it possible to keep even large databases completely in main memory. Hence, efficient searching techniques for main memory databases have become important in many applications. Unlike disk based databases where the main goal of searching is to minimize disk page accesses, in main memory databases the primary focus is to reduce the number of complex distance computations. For metric spaces, indexing based on vantage points is one of the popular approaches for implementing range queries. The basic idea behind vantage point based transformation is to express each data point (or feature vector) in terms of its distances from the precomputed set of points called vantage points (also known as reference points or pivots). Range query in original space is then implemented as a minimum bounding box query in the transformed space (also known as vantage space or pivot space). Selection of vantage points is a critical step in this scheme. However, to the best of our knowledge all the existing work in this area focuses on using data points as vantage points. The main motivation behind this strategy is that vantage points selected from the dataset will closely follow the properties of the data distribution. However, this strategy also has some serious drawbacks.

- Time complexity of vantage point selection algorithm is directly dependent on the size of the database. This can be a serious problem when selecting vantage points in a large database.

- This requires the database to be static. All the data is required to be available at the time of selecting the vantage points.

- You need to reselect the vantage point for each different database even though it may have the same properties (number of dimensions, distribution etc.).

We address these issues by proposing a novel scheme for vantage point selection that is independent of the data. The only requirement of the proposed scheme is that data space should be bounded. In other words, each dimension of a feature vector can take a real value within a certain range. Without loss of generality, we assume that the data is bounded in the unit (hyper)cube. The proposed algorithm depends only on the dimensionality of the space and the number of vantage points. Vantage point selection being independent of the data, allows dynamic insertions, deletions and updates to the database without having to change the vantage points.

6.1 Vantage point transformation basics

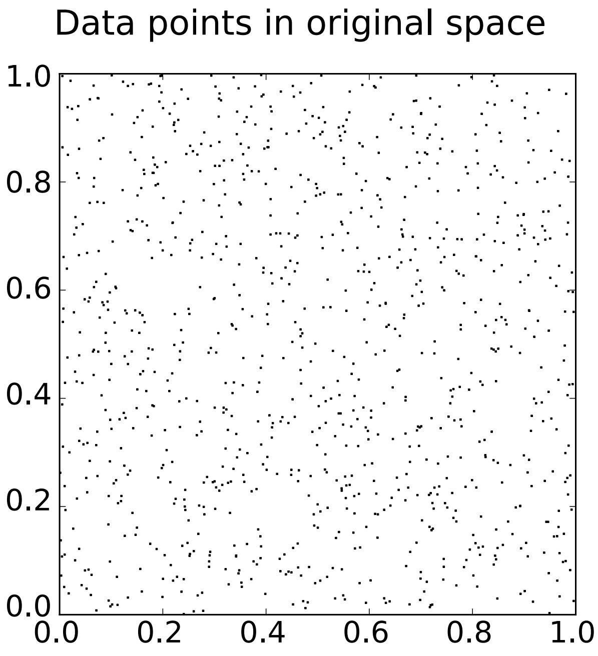
We begin the discussion by first formally presenting data transformation and query transformation using vantage points and then move on to the various theories that drive our vantage point selection algorithm. We will use euclidean distance throughout this discussion unless explicitly specified otherwise.

6.1.1 Data transformation

As always, we denote the set of data points in n dimensional euclidean space as $\mathbf{D}$. Let $p = (p_1, p_2, \ldots p_n) \in \mathbf{D}$ be a feature vector representing a particular object. Let $\mathbf{V} = \{\mathbf{v_1}, \mathbf{v_2}, \ldots \mathbf{v_m}\}$ be a set of m vantage points. Each point p is then transformed to point $\hat{p}$ such that, $\hat{p} = (\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_m)$ where, $\hat{p}_i$ is distance of p from the vantage point $\mathbf{v_i}$, i.e.,

$$\begin{aligned}\hat{p}_i &= d(p, \mathbf{v_i}) \\ &= L_2(p, \mathbf{v_i}) \quad \text{since euclidean distance is used as the distance measure}\end{aligned}$$

Figure 6.1 shows how the points in 2-D space are mapped in the transformed space. Here, we assume that the number of vantage points $m = 2$. Further, we use $\mathbf{v_1} = (0, 0)$ and $\mathbf{v_2} = (0, 1)$ as the two vantage points. This transformation, in general, is not unique, i.e., multiple points in the original space can map into the same point in the transformed space. As an example, consider $\mathbf{v_1} = (0, 0)$ and $\mathbf{v_2} = (1, 1)$ as vantage points and $p = (0.4, 0.6)$ and $q = (0.6, 0.4)$ as two data points. It can be easily verified that $\hat{p} = \hat{q} = (0.72, 0.72)$. We define this event as a "collision". In the later sections, we propose heuristics aimed at eliminating or reducing the number of collisions.



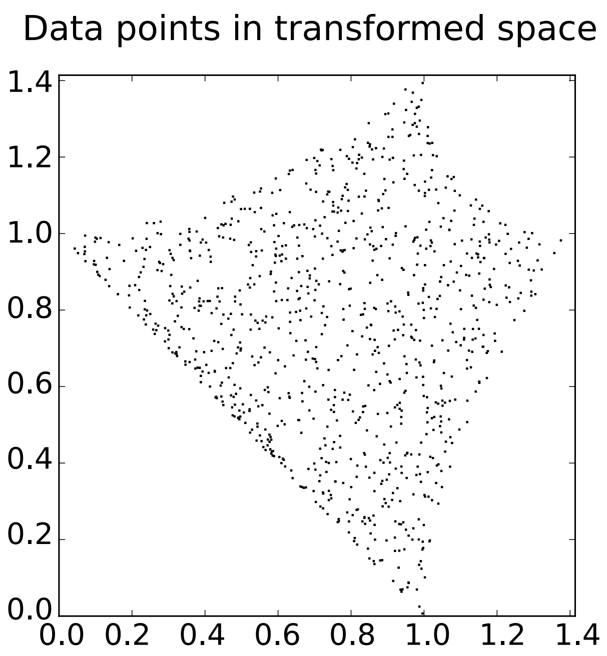


Figure 6.1: Transformation of points in 2-D space

6.1.2 Query transformation

In a vantage point based index, a range query in the original space is mapped into a bounding box query in the transformed space. We illustrate this mapping using the example in figure 6.2. Consider a range query (in euclidean distance) at a point $(0.2, 0.7)$ with radius 0.1. All the points inside the circle of radius 0.1 satisfy the query. Consider the same two vantage points $\mathbf{v_1} = (0, 0)$ and $\mathbf{v_2} = (0, 1)$ for this example. The center of the query then maps to $(0.73, 0.36)$ in the transformed space. All the points inside the query space (circle) map roughly to the shaded region in the transformed space. With respect to $\mathbf{v_1}$, the minimum and maximum distances of any point in the query are 0.63 (dx_1) and 0.83 (dx_2) respectively. Similarly, with respect to $\mathbf{v_2}$, the minimum and maximum distances of any point in the query are 0.26 (dy_1) and 0.46 (dy_2) respectively. As can be seen from the illustration, the minimum bounding box query $([0.63, 0.83], [0.26, 0.46])$ in the transformed space contains all the points of the range query in the original space. Thus, a range query is now mapped

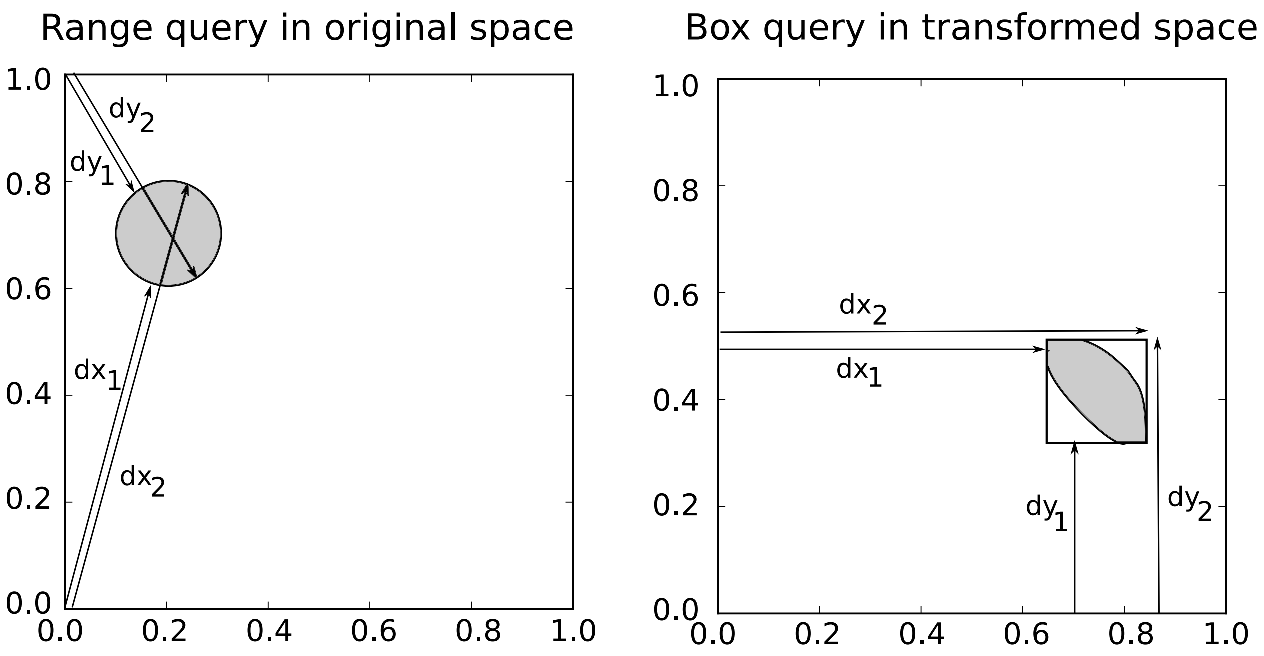
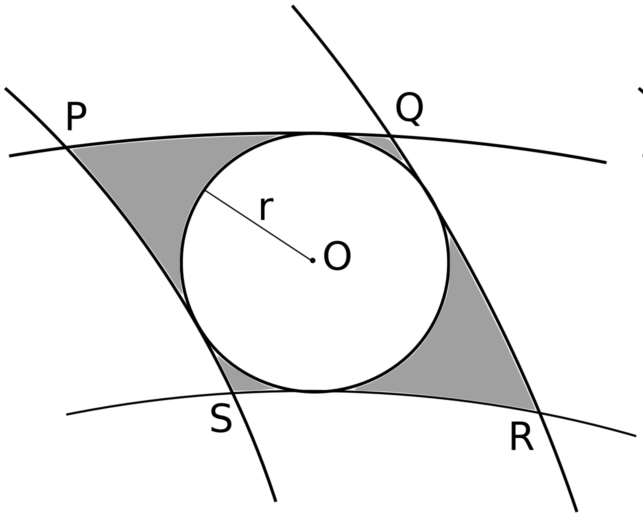
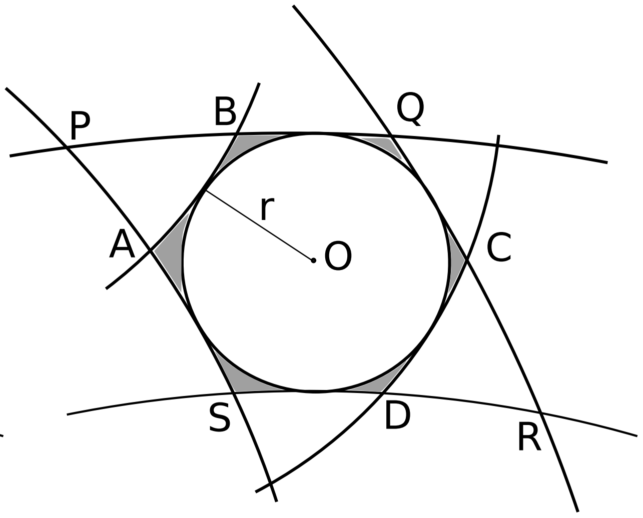


Figure 6.2: Mapping range query to box query through space transformation

to a minimum bounding box query. Note that the mapping is not exact in the sense that the minimum bounding box query has many more points than those inside the range query. As we will see later, the amount of false positives depends on the relative position of the vantage points. In fact, if one of the vantage points is the query center, then the box query is the exact mapping of the range query with no false positives. However, the main point here is that the box query is always a superset of the range query. At a first glance, it may appear that the amount of false positives in the transformed space is much more than that in the bounding box in the original space. However, it should be remembered that by increasing the number of vantage points, we can reduce the number of false positives. Figure 6.3a shows visualization of a box query using two vantage points in the original space. The shaded region corresponds to the amount of false positives. Figure 6.3b shows visualization of a box query using three vantage points. As can be seen, adding one more vantage point significantly reduces the amount of false positives. This potential for false positive reduction



(a) Box query with 2 vantage points



(b) Box query with 3 vantage points

Figure 6.3: Reduction in false positives with increasing number of vantage points

makes box queries using vantage points particularly attractive over bounding box queries in the original space.

6.2 Selection of vantage points in 2-D space

In this section, we develop some theorems that will be the basis for vantage point selection in 2-D space. Number of collisions and the number of false positives are some of the important aspects of vantage point selection that need attention. We will consider each of them one by one.

6.2.1 Minimizing Collisions

Following theorem gives vantage points that avoid mapping multiple points in the original space into the same point in the transformed space, thus avoiding collisions. This is a one to one functional mapping.

Theorem 6.2.1. *Assuming that all the data points are in the unit square, if the two vantage points lie on an axis of the 2-D space, then the resulting vantage point based transformation function is one to one.*

Proof 6.2.1. *We prove the theorem when both the vantage points are on the y-axis. The proof for vantage points on x-axis is similar. Let* $(0, y_1)$ *and* $(0, y_2)$ *be the two vantage points. Let* (x, y) *be a point in the original space whose vantage point based functional mapping is* (d_1, d_2)*. Thus, the distance of the point* (x, y) *from the vantage point* $(0, y_1)$ *is* d_1 *and that from the vantage point* $(0, y_2)$ *is* d_2*. In order to prove that this mapping is collision free, we show that, given* (d_1, d_2)*, we can uniquely calculate the point* (x, y)*. Hence, we have to find the solutions to the following two equations :*

$$x^2 + (y - y_1)^2 = d_1^2 \tag{6.1}$$

$$x^2 + (y - y_2)^2 = d_2^2 \tag{6.2}$$

Subtracting equation 6.2 from equation 6.1 we get,

$$\therefore -2yy_1 + y_1^2 + 2yy_2 - y_2^2 = d_1^2 - d_2^2$$

$$\therefore 2y(y_2 - y_1) = d_1^2 - d_2^2 - y_1^2 + y_2^2$$

$$y = \frac{d_1^2 - d_2^2 - y_1^2 + y_2^2}{2(y_2 - y_1)}$$

$$\Rightarrow x = \pm\sqrt{d_1^2 - (y - y_1)^2} \tag{6.3}$$

From equation 6.3, whenever x is real, one of the two values of x is negative. As all the data are in the first quadrant of the space, there remains exactly one valid solution for the

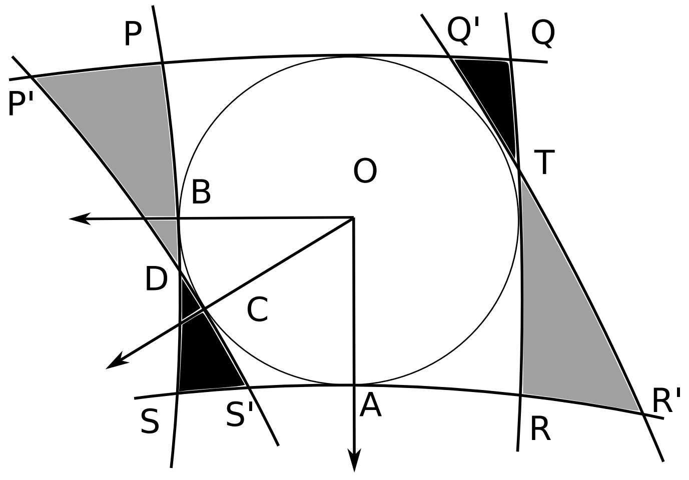


Figure 6.4: Minimizing the false positives

equations 6.1 and 6.2. Hence, given (d_1, d_2)*, one can always calculate a unique corresponding point* (x, y)*, in other words, the transformation is collision-free.*

6.2.2 Reduction of false positives

As described in section 6.1, vantage point based approach has to handle false positives. The amount of false positives is dependent on the angle subscribed by the two vantage points at the query center. As this angle gets closer to $\pi/2$, the false positive percentage gets smaller. We will justify this statement by starting with two vantage points that make an angle of $\pi/2$ at the query center and then showing that the number of false positives increases as we move (everywhere in the space) one of the vantage points such that the angle becomes either less or more than $\pi/2$. Figure 6.4 shows a query with center at O. The arcs are created based on the nearest and the farthest distance of any point within the query space from the vantage points as discussed in section 6.1. As shown in the figure, arcs SAR and $PQ'Q$ correspond to minimum and maximum distances from the first vantage point (i.e., $\mathbf{v_1}$ lying on ray OA). Arcs PBS and QTR correspond to minimum and maximum distances from the second vantage point (i.e., $\mathbf{v_2}$ lying on ray OB). Angle subscribed by the vantage points $\mathbf{v_1}$ and $\mathbf{v_2}$ at the query center O is $\pi/2$. In this case, the query space is the squarish region bounded by $PQRS$. Now assume that the second vantage point $\mathbf{v_2}$ moves such that it now lies along ray OC and angle subscribed by the two vantage points at O is now slightly less than $\pi/2$. After this change, the query region is the rhomboidal shape bounded by $P'Q'R'S'$. Note that the two regions $P'PD$ and $R'RT$ got added (shaded gray) to the query space and the two regions $SS'D$ and $Q'QT$ got removed (shaded black) from the query space. We observe that whenever, $r << d_1, d_2$, the sum of the areas $P'PD$ and $RR'T$ is greater than the sum of the areas $QQ'T$ and $SS'D$. Hence, the amount of false positives increases. The following theorem formally states and proves the limiting case when $d_1, d_2 \to \infty$:

Theorem 6.2.2. *Let $\mathbf{v_1}$ and $\mathbf{v_2}$ be two vantage points and q be a query center such that $d(\mathbf{v_1}, q) = d_1$ and $d(\mathbf{v_2}, q) = d_2$. Let θ be the angle subscribed by points $\mathbf{v_1}$ and $\mathbf{v_2}$ at the query center q. As $d_1, d_2 \to \infty$ the amount of false positives $E(\theta)$ is minimum when $\theta = \pi/2$.*

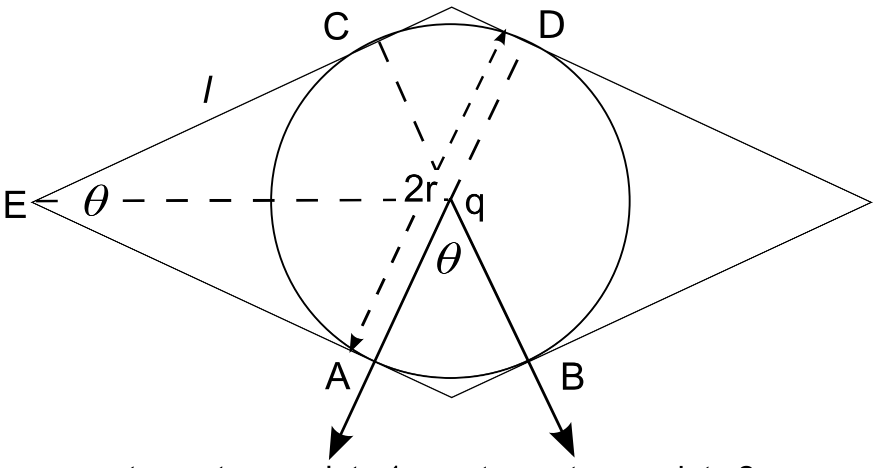


Figure 6.5: Proof of theorem 6.2.2

Proof 6.2.2. *We first derive an expression for percentage of false positives in terms of angle θ. Since, $d_1, d_2 \to \theta$ the arcs enclosing the query circle can be considered to be straight lines.*

Hence, the circle is effectively enclosed in two sets of parallel lines (i.e. a parallelogram). Geometrical properties of circles can be used to prove that the parallelogram is in fact a rhombus. So we are approximating the circular query space with a rhomboidal shape. As shown in the figure, the distance between the parallel lines (i.e. height of the rhombus) is equal to the diameter of the circle = $2r$. Also, internal angles of the rhombus are θ and $(\pi - \theta)$. Based of this information, it can be seen that,

$$\sin\theta = \frac{2r}{l} \text{ where, } l \text{ is the side of the rhombus}$$
$$\Rightarrow l = \frac{2r}{\sin\theta}$$

Hence, area of the rhombus is,

$$\begin{aligned} A_{rhombus} &= 2rl \\ &= \frac{4r^2}{\sin^2\theta} \end{aligned}$$

For uniformly distributed data, the amount of false positives FP can be estimated as,

$$\begin{aligned} FP &= \frac{A_{rhombus} - A_{circle}}{A_{circle}} \\ &= \frac{4}{\pi \sin^2\theta} - 1 \end{aligned} \tag{6.4}$$

It can be seen that equation 6.4 attains the minimum value for $\theta = \pi/2$ i.e. when the distance vector OA and OB are perpendicular. Theorem 6.2.2 follows directly from equation 6.4.

Because points lying on the axis gives unique mapping and $\pi/2$ condition gives less false positive, we use these two criteria as the basis of our vantage point selection technique.

6.2.3 Corner points as vantage points

Based on theorem 6.2.1, we select the vantage points on an axis of the space. The following proposition highlights an important property of the angles subscribed by the vantage points on y-axis at the query point.

Proposition 6.2.1. *As the two vantage points on the y-axis move farther from each other the angle subscribed by these two points at a query point increases.*

Proof follows directly from the figure 6.6.

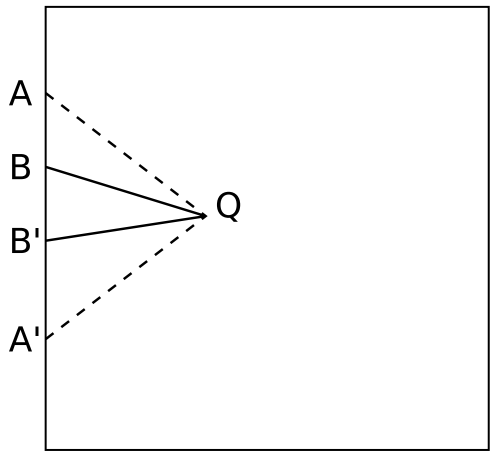


Figure 6.6: Angle subscribed by AA' at Q is larger than the one by BB'

When the two vantage points are infinitesimally close to each other, any point in the space will form a zero degree angle with the vantage points. As the two vantage points move away from each other (refer figure 6.6), the angles they form at any query point in the space gradually increase. Hence the expected angle also increases as the vantage points move away from each other.

Our next theorem proves that the number of points, which has angles with respect to two vantage points closer to $\pi/2$, is maximum when the vantage points are two adjacent corner points in the space (i.e., two extreme points on an axis). We state the theorem below for corner points along the y-axis (i.e., $(0,0)$ and $(0,1)$). Theorems and the corresponding proofs for any one of the other three pairs of adjacent corner points are similar.

Proposition 6.2.2. *Let $(0,y)$ and $(0,y+y')$ be the two vantage points on the y-axis. Then the number of query points, that can subscribe angles in the range $[\pi/2-\delta\theta,\pi/2+\delta\theta]$ and $0\le\delta\theta\le\pi/2$ with the two vantage points, increases with increasing y'.*

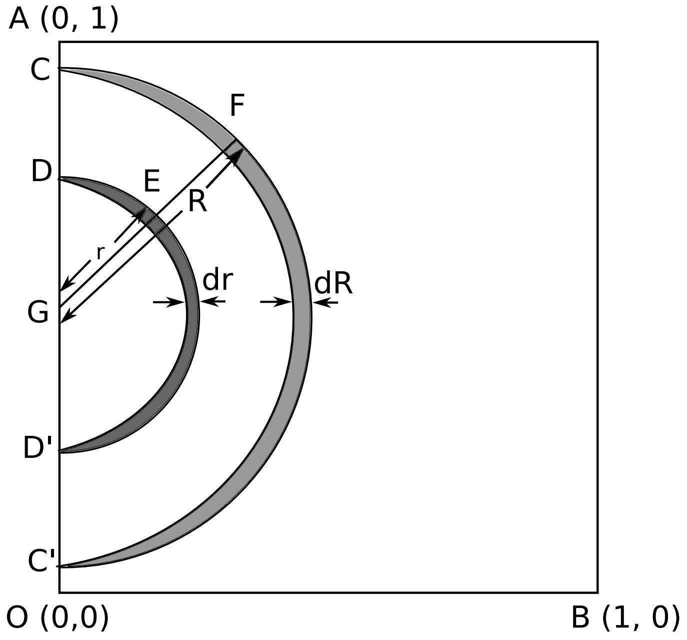


Figure 6.7: Proof of theorem 6.2.2

Justification. *We consider two sets of candidate vantage points $\{D,D'\}$ and $\{C,C'\}$ as shown in figure 6.7. Without loss of generality, we assume that $G=(0,0.5)$ is the mid-point of both DD' and CC'. The larger semi-circle (with radius R) corresponds to the locus of points subscribing an angle $\pi/2$ with the vantage points C,C' and the smaller semi-circle (with radius r) corresponds to the points making an angle $\pi/2$ with the vantage points D,D'. Due to our assumption that G is the mid point, the two semi-circles are concentric. Now connect the center G with any point F on the larger semi-circle. This line segment intersects* *smaller semi-circle at E. $\angle CFC' = \angle DED' = \pi/2$ and $\angle CGC' = \angle DGD' = \pi$. Angle subscribed by a query point lying along segment FG with respect to vantage points C, C' increases from $\pi/2$ to π as the point moves from F toward G. Similarly, angle subscribed by a query point lying along segment EG with respect to vantage points D, D' increases from $\pi/2$ to π as the point moves from E toward G. Since length(FG) > length(EG), the rate at which angles increase along ray EG (with respect the points D, D') for smaller semi-circle is larger than the rate at which angles increase along ray FG (with respect to C, C'. Consider a small strip around larger semi-circle which corresponds to the points making angles in the range $[\pi/2, \pi/2 + \delta\theta]$ with respect to C, C'. Draw a similar strip along the smaller semi-circle for angles with respect to D, D'. Due to smaller rate of increase in angles, for a given $\delta\theta$, $dR > dr$, i.e., the strip will be thicker for larger semi-circle than corresponding strip for a smaller semi-circle. Because the circumference of the semi-circles with radius R is bigger than the circumference of the semi-circle with radius r, the area of the strip around larger semi-circle is larger than the corresponding strip around smaller semi-circle. For uniformly distributed data, this area is directly proportional to the number of points that can make angles $[\pi/2, \pi/2 + \delta\theta]$.*

Now consider those points that are lying outside the larger semi-circle. Three facts can be stated for these points.

- *They always subscribe an angle less than $\pi/2$ with either of the two vantage point sets.*
- *For a given point in this space, the angle subscribed with respect to CC' is larger than that subscribed with respect to DD'.*
- *The area of the space outside of the bigger semi-circle is larger than the area of the space inside of the semi-circle as long as the points CC' are within the corner points.*

Based on the above three points we conclude that for points outside the bigger semicircle, more of these points will subscribe angles with respect to CC' that are closer to $\pi/2$ than those angles with respect to DD'.

Following theorem proves that the expected angle is $\pi/2$ when the two vantage points selected are the corner points.

Theorem 6.2.3. *If* $\mathbf{v_1} = (0, 0)$ *and* $\mathbf{v_2} = (0, 1)$ *are the two vantage points then the expected value of the angle at a query point subscribed by the two vantage points is* $\pi/2$.

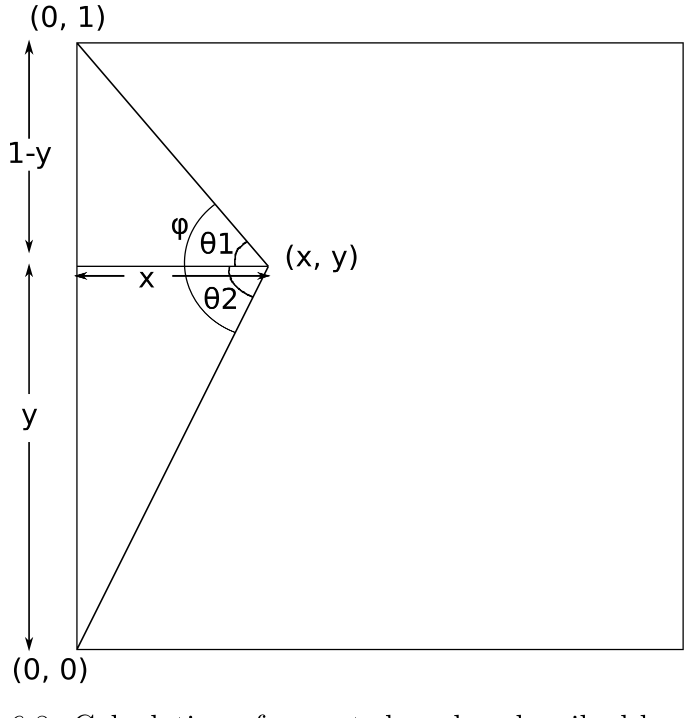


Figure 6.8: Calculation of expected angle subscribed by a point

Proof 6.2.3. *As shown in the figure, let* (x, y) *be a random point in the unit square. The angle* ϕ *subscribed by this point with respect to the two vantage points can be calculated as* *the sum of the two angles θ_1 and θ_2.*

$$\begin{aligned}
\phi = \theta_1 + \theta_2 &= \tan^{-1}\left(\frac{1-y}{x}\right) + \tan^{-1}\left(\frac{y}{x}\right) \\
\therefore E(\phi) &= \frac{\int_0^1 \int_0^1 \phi \; dx \; dy}{\int_0^1 \int_0^1 dx \; dy} \\
&= \int_0^1 \int_0^1 \tan^{-1}\left(\frac{1-y}{x}\right) + \tan^{-1}\left(\frac{y}{x}\right) dx \; dy \\
&= \frac{1}{2}\int_0^1 \Bigg[y \log\left(1 + \frac{1}{y^2}\right) - (y-1)\log\left(1 + \frac{1}{(y-1)^2}\right) \\
&\qquad + 2\tan^{-1}(1-y) + 2\tan^{-1}(y) \Bigg] dy \\
&= \pi/2
\end{aligned}$$

Theorem 6.2.3 combined with proposition 6.2.1 shows that as the vantage points move away from each other, expected angle of a query point increases. Theorem 6.2.2 shows, that the number of query points forming an angle close to $\pi/2$ increases as the vantage point move father apart. Based on all three results, we can conclude that when the vantage points are close to each other, the angles formed by the vantage points with the query points are skewed toward zero degrees with mean less than $\pi/2$. On the other hand when the vantage points are end points of the y-axis, the angles are more or less balanced around the mean of $\pi/2$. We propose the following conjecture that captures these ideas.

Conjecture 6.2.4. *As the vantage points on y-axis move farther and farther from each other, the second moment about $\pi/2$, $E[(\theta - \pi/2)^2]$, of the angles decreases.*

In other words, as the vantage points on y-axis move farther and farther from each other, more and more angles come closer to $\pi/2$. Tables 6.1 and 6.2 show empirical results obtained (using 10,000 random query points) for various possible vantage points on y-axis.

For ease of interpretation, the values are given in degrees instead of radians. Labels "Mean" and "Moment$_{90}$" are used to refer to average angle and second moment of angles about 90° respectively. In table 6.1, $\mathbf{v_1}$ is stationary and $\mathbf{v_2}$ is gradually moves away from $\mathbf{v_1}$. In table 6.2, both the points are initially close to the center of the axis and move gradually away from each other. Empirical results in these tables justify the conjecture. Based on the statements of section 6.2.2, this directly translates to lesser false positive percentage. All the theorems and the conjecture presented so far will provide some intuitive basis for our vantage point selection algorithm in higher dimensions.

Table 6.1: Angle statistics for various vantage points on y-axis

$\mathbf{v_1}$	$\mathbf{v_2}$	**Mean**	**Moment$_{90}$**
(0, 0)	(0, 0.1)	7.31	6947.03
(0, 0)	(0, 0.2)	15.78	5834.77
(0, 0)	(0, 0.3)	25.12	4812.36
(0, 0)	(0, 0.4)	34.85	3898.39
(0, 0)	(0, 0.5)	44.62	3110.59
(0, 0)	(0, 0.6)	54.47	2427.74
(0, 0)	(0, 0.7)	64.39	1893.86
(0, 0)	(0, 0.8)	73.78	1506.00
(0, 0)	(0, 0.9)	82.42	1271.09
(0, 0)	(0, 1.0)	90.06	1191.51

Table 6.2: Angle statistics for various vantage points on y-axis

$\mathbf{v_1}$	$\mathbf{v_2}$	**Mean**	**Moment$_{90}$**
(0, 0.45)	(0, 0.55)	10.02	6526.42
(0, 0.4)	(0, 0.6)	19.91	5256.3
(0, 0.35)	(0, 0.65)	29.71	4217.90
(0, 0.3)	(0, 0.7)	39.38	3366.15
(0, 0.25)	(0, 0.75)	48.85	2675.91
(0, 0.2)	(0, 0.8)	58.06	2137.62
(0, 0.15)	(0, 0.85)	66.97	1731.64
(0, 0.1)	(0, 0.9)	75.52	1449.36
(0, 0.05)	(0, 0.95)	83.52	1285.47
(0, 0)	(0, 1.0)	90.63	1244.09

6.2.4 Choosing more than two vantage points

The discussion so far focuses on choice of first two vantage points. But it is possible to use more than two vantage points. An apparent disadvantage of this is the increased dimensionality (and the memory requirement) of the index. However, increasing vantage points also decreases false positives and thus decreases the number of distance computations. In 2-D space, there are four corner points. Once the first two vantage points (i.e. $(0, 0)$ and $(0, 1)$ are chosen, the other two corner points can be chosen in any order as the third and the fourth vantage points. Our experiments show that the order does not make any difference. The figure 6.9 shows the effect of increasing the number of vantage points. As is seen from the figure, using 3 or 4 vantage points provide slightly better results.

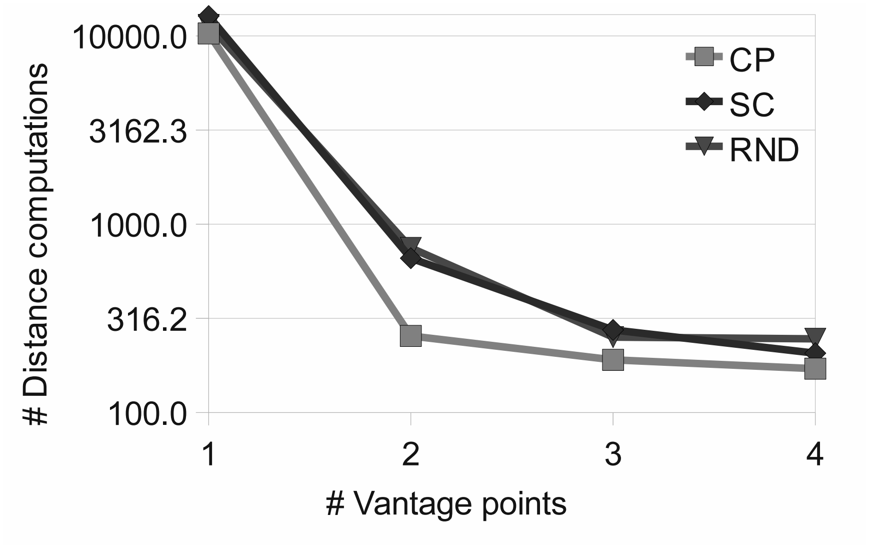


Figure 6.9: Effect of increasing the number of vantage points

6.2.5 Experimental results with 2D data

Here, we present some preliminary experimental results using the proposed heuristics for selecting vantage points. Two dimensional uniformly distributed synthetic data is used for these experiments. We use 2 and 3 vantage points for each of the experiments. Labels CP-2 and CP-3 refer to corner points method with two and three vantage points, respectively. Labels SC-2 and SC-3 refer to spacing correlation based method [62] while labels RND-2 and RND-3 refers to random vantage points. Figure 6.10 shows effect of increasing database size while the figure 6.11 shows the effect of increasing query radius. It can be seen that corner point based approach consistently provides better results. Further, increasing the number of vantage points also decreases the distance computations further. This highlights effectiveness of the proposed scheme in practice. More results comparing our methods with others for several real and synthetic databases are given in section 6.4

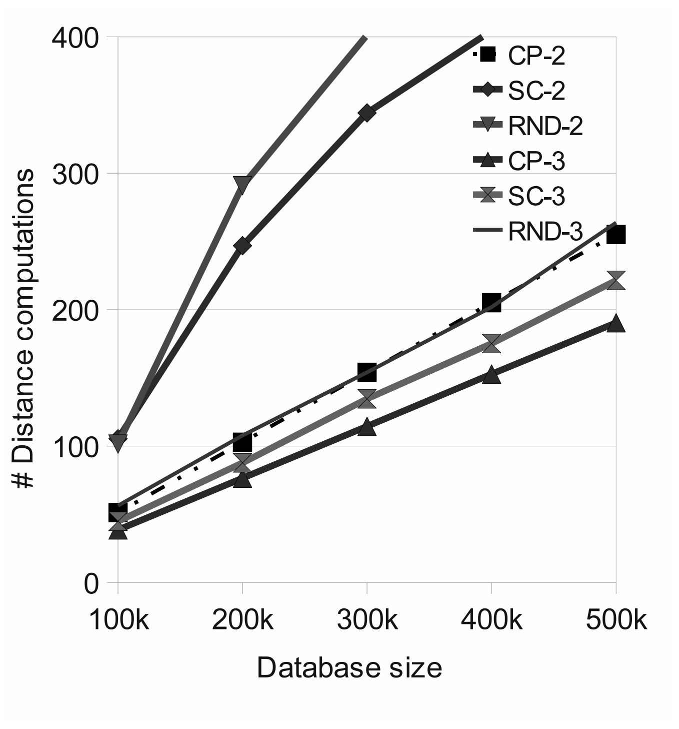


Figure 6.10: Effect of increasing database size

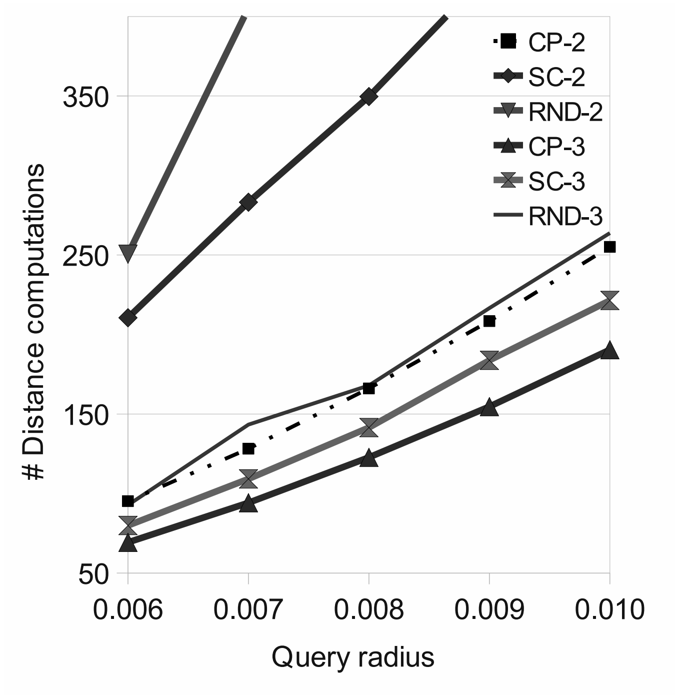


Figure 6.11: Effect of increasing query radius

6.3 Selection of vantage points in higher dimensional spaces

We now extend the theories developed so far for higher dimensional data spaces. As shown in figure 6.3, in 2-D space, each additional vantage point reduces more false positives. The space bounds corresponding to each vantage points are arcs of two concentric circles in 2-D. In higher dimensions however, the space bounds are in fact concentric hyper-spherical surfaces. This makes analysis of higher dimensional spaces more challenging. In the following sections, we demonstrate that the concept of corner points indeed works even for higher dimensions. We also extend the heuristics in 2-D spaces to guide the choice of vantage points in higher dimensional spaces.

6.3.1 Effectiveness of data independent vantage points

In this section, we demonstrate the effectiveness of corner point approach for selecting data independent vantage points. Although data independent vantage points may overcome some of the important limitations of the data dependent method, they are useful if and only if they can provide comparable (or better) performance in terms of the number of distance calculations. Van Leuken et. al. [62] have proposed two metrics namely spacing variance and pairwise distance correlation for evaluating the goodness of a set of data points as vantage points. Spacing between two consecutive (with respect to distance from vantage point $\mathbf{v}$) points p and q is $d(q, \mathbf{v}) - d(p, \mathbf{v})$. Let μ be the average spacing. The spacing variance is formally defined as,

$$\sigma_{sp}^2 = \frac{1}{N-1} \sum_{p \in \mathbf{D},\ q \in \mathbf{D}} [(d(q, \mathbf{v}) - d(p, \mathbf{v})) - \mu]^2$$

where, N is the number of points in the database. Distance correlation is the correlation between two sequences of distances where each sequence consists of the distances of the data points from a vantage point. Formally let $\hat{d}_{i,1}, \hat{d}_{i,2} \ldots, \hat{d}_{i,N}$ be distances of N points from vantage point $\mathbf{v_i}$ then distance correlation for vantage points $\mathbf{v_i}$ and $\mathbf{v_j}$ is,

$$C(\mathbf{v_i}, \mathbf{v_j}) = \frac{\sum_{k=1}^{N} (\hat{d}_{i,k} . \hat{d}_{j,k}) - \sum_{k=1}^{N} \hat{d}_{i,k} \sum_{k=1}^{N} \hat{d}_{j,k}}{\sqrt{N \sum_{k=1}^{N} \hat{d}_{i,k}^2 - (\sum_{k=1}^{N} \hat{d}_{i,k})^2} \sqrt{N \sum_{k=1}^{N} \hat{d}_{j,k}^2 - (\sum_{k=1}^{N} \hat{d}_{j,k})^2}}$$

They propose that choosing vantage points with lower spacing variance ensures that the database objects are uniformly distributed in the transformed space. Choosing vantage

Table 6.3: Average Spacing Standard Deviation for some datasets

Dataset Type	$\sigma_{\mathbf{sp}}$**for DP**	$\sigma_{\mathbf{sp}}$ **for CP**
Synthetic Uniform	0.00093	0.00033
Weather Data	0.00022	0.00018
Audio feature data	0.00027	0.00019

Table 6.4: Top 3 correlation coefficients for some datasets

Dataset Type	**Corr-coeff. for DP**	**Corr-coeff. for CP**
Synthetic Uniform	0.55, 0.51, 0.43	0.42, 0.32, 0.32
Weather Data	0.99, 0.98, 0.98	0.94, 0.90, 0.86
Audio feature data	0.97, 0.87, 0.81	0.84, 0.75, 0.73

points which have lower distance correlation guarantees that no vantage point is redundant. We perform an empirical study in which vantage points are selected as a random sample of data points and as a random sample of corner points (i.e., extreme points in each axis of the space). We then compare the two methods in terms of standard deviation of spacing and distance correlation coefficients. Tables 6.3 and 6.4 demonstrate our findings. In the tables, label DP corresponds to using Data Points as vantage points and the label CP corresponds to using Corner Points as vantage points. We used one synthetic data set of uniformly distributed points and two real datasets (more description about the datasets is given in section 6.4). Table 6.3 shows the average of the standard deviations of the distances over all vantage points for the two methods. It can be seen that for all three datasets, using corner points as vantage points gives much smaller standard deviation than using data points as vantage point. Table 6.4 shows the top 3 correlation coefficients. It can be seen that corner points approach consistently results in lower correlation coefficients. We will use more effective heuristics for choosing the vantage points for both methods in the next section. These results clearly suggest that corner points may be a better choice for vantage points.

The main challenge in using corner points as data independent vantage points is that different data may have different distributions. Hence, in order to choose a data independent set of vantage points, we need to ensure that it can provide low spacing variance and low distance correlation, irrespective of the data that are being indexed. In the following subsections, we develop heuristics that focus on these objectives.

6.3.2 Minimum Distance Variance

The measures proposed by Van Leuken et al. are data dependent, i.e., one can evaluate goodness of a set of vantage points only with respect to the data that are to be indexed. However, for data independent choice of vantage point selection one can no longer use any information of the dataset. The key idea is to ensure that the vantage points are spread uniformly throughout the space. That will guarantee that each vantage point gets a view of the dataset from a different perspective thus resulting in lower distance correlation. We use variance of the pairwise-distance among vantage points as the measure of spread of vantage points. If the vantage points are spread uniformly in the space then their distance from each other will be more or less the same. In other words, the pairwise-distance variance will be minimum. Minimizing pairwise distance variance alone may not be sufficient to ensure low correlation among the vantage points. As mentioned in [16, 62], it is important that the vantage points are neither too close nor too far from each other. We do not want to end up with vantage points which are all at the distance 1 (minimum) or at the distance $\sqrt{n}$ (maximum). Further, the number of such points which are at these distances are small. So as the number of vantage points increases, we will invariably be forced to choose the points that result in higher pairwise-distance variance. We focus on choosing vantage points that

Table 6.5: Average pairwise distance between corner points

#Dimensions (n)	$\sqrt{\frac{n}{2}}$	**Avg. distance**	**% difference**
2	1.00	0.85	14.64%
4	1.41	1.34	5.36%
6	1.73	1.68	2.82%
8	2.00	1.96	1.88%
10	2.24	2.20	1.42%

are all at a distance of $\sqrt{\frac{n}{2}}$ with each other. This is based on the fact that average of the distances from a corner point to all other corner points is very close to $\sqrt{\frac{n}{2}}$. This is verified by the result given in Table 6.5 which compares average distance between corner points in n dimensional space with $\sqrt{\frac{n}{2}}$. It can be seen that as the number of dimensions increases, the two values converge. Following section provides a heuristic to create a set of vantage points that minimizes the distance variance.

6.3.2.1 A heuristic for selecting vantage points

We can get a good set of vantage points by starting with corner points that are at a distance $\sqrt{\frac{n}{2}}$, and repeatedly adding points in such a way that the pairwise distance variance is as small as possible. The heuristic algorithm for choosing vantage points is presented in Algorithm 2. We begin with origin as the first vantage point. The second vantage point is obtained by flipping half the dimensions of origin to 1. This guarantees that the initial set of vantage points is always at the distance of $\sqrt{\frac{n}{2}}$. In each iteration, we go through each of the corner points (that are not yet selected as vantage point) and find the minimum pairwise-distance variance if that point is added to the set of vantage points (in the algorithm, the function GetDistanceVariance() returns the pairwise-distance variance of a set of points). The point whose addition results in the smallest pairwise-distance variance is chosen as the next vantage point. We continue the process until the desired number of vantage points are obtained. Note that our algorithm is greedy in the sense that at each step it chooses the current best point. However, it may not result in a vantage point set which has globally minimum pairwise-distance variance. Another important point to note is that even though our algorithm begins with origin as the first point, one may begin with any corner point as the first vantage point.

Algorithm 2 Generate m vantage points for n dimensional unit hypercube

```
Input: Number of dimensions $n$ and number of vantage points $m$.
Output: A set of $m$ vantage points all of which are at a distance of $\sqrt{\frac{n}{2}}$ from each other.
Algorithm
 1: Vantage point set $V \leftarrow \phi$
 2: Add origin to vantage point set.
    $V = V \cup \{a\ string\ of\ n\ 0s\}$
 3: Next vantage point is obtained by flipping first $n/2$ dimensions.
    $V = V \cup \{a\ string\ of\ n/2\ 1s\ followed\ by\ n/2\ 0s\}$
 4: for $i = 3$ to $m$ do
 5:    minDistVariance $\leftarrow \infty$, nextPoint $\leftarrow \phi$
 6:    for each corner point $p$ not yet selected do
 7:       $V_{temp} = V \cup \{p\}$
 8:       distVariance = GetDistanceVariance($V_{temp}$)
 9:       if distVariance < minDistVariance then
10:          minDistVariance = distVariance
11:          nextPoint = $\{p\}$
12:       end if
13:    end for
14:    $V = V \cup$ nextPoint
15: end for
16: return $V$
```

The number of corner points in an n dimensional space is 2^n. Hence, the for-loop in step 6 in the algorithm has complexity of $O(2^n)$. This loop is repeated $(m-2)$ times (due to the for-loop in step 4). Step 8 of the algorithm can be implemented in an incremental fashion such that it runs in $O(m)$ time. Thus, the overall complexity of this algorithm is $O(m(2^n + m))$. Thus, the time complexity of the algorithm is exponential in the number of dimensions and polynomial in the number of vantage points. The exponential time complexity may make the algorithm seem undesirable when the number of dimensions is very high. However, we would like to highlight that this algorithm needs to be run exactly once for each value of n. Once the set of vantage points (essentially the order in which corner points are chosen as vantage points) is established, the set can be reused for any other application. We will revisit this issue of complexity later in this chapter.

6.3.3 Index structure and query algorithm

We now present the index structure and the query algorithm used for our experiments. A standard KD-tree is used to index the transformed feature vectors. A set of vantage points is first generated before beginning to construct the index. For each feature vector, we transform it by using the distances from the vantage points and then insert the transformed feature vector into the KD-tree.

For a range query with query center at point o and radius r, we first derive the transformed query center $\hat{o} = (\hat{o}_1, \hat{o}_2, , \dots, \hat{o}_m)$. The corresponding bounding box query is then computed as $([\hat{o}_1 - r, \hat{o}_1 + r], [\hat{o}_2 - r, \hat{o}_2 + r], \dots, [\hat{o}_m - r, \hat{o}_m + r])$. The box query is executed using the standard KD-tree box query execution algorithm [9]. For each node of the KD-tree satisfying the box query, its actual distance is computed to eliminate all the false positives.

6.4 Experimental results

In this section we present experimental results demonstrating effectiveness of the vantage points generated by the proposed vantage point generation scheme. We begin with detailed description of the experimental set-up and the databases used.

6.4.1 Experimental set-up and datasets used

We use three synthetic databases each with different distribution and six real databases each coming from a different application domain. The first synthetic database contains 10-dimensional uniformly distributed data. The second synthetic database contains 10-dimensional skewed data which follows exponential distribution. Our third synthetic database is a 10-dimensional data containing a mixture of Gaussian clusters with random means and covariance matrices.

We use two 2 dimensional and four high dimensional real databases for our testing. The first 2-D database is a GIS database which consists of coordinate points obtained through GPS. The second 2-D database comes from the field of computational fluid dynamics. It consists of collection of points which are used to model the air flows over and around aerospace vehicle [46, 48]. The first high dimensional real database is the same as the one used for experiments in previous chapters (sections 3.7, 3.8.3 and 4.3.4). It consists of image feature vector data. For each image, first three moments for hue, saturation and intensity are used as the image feature vector (similar to the features used in [59]). Thus, each data point is a 9-dimensional feature vector. The second real database contains 12-dimensional music feature vectors [10]. Each feature vector is essentially a 12 dimensional timbre average of the music sequence. Our third real database contains human speech feature vectors. Each speech segment is divided into a number of windows of equal width. For each segment, 8 features such as mean frequency, percentage silence etc. are calculated. The last real database contains 12-dimensional weather information [1] such as average temperature, pressure, wind speeds.

All the experiments are run on 2.7 GHz quad-core ((512KB cache) machines running

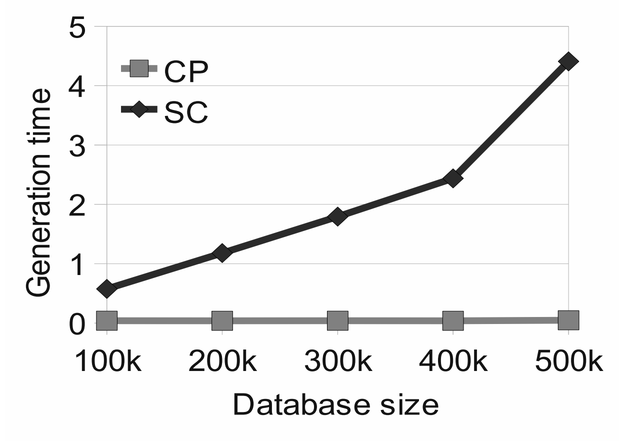
Linux. For all the experiments, we compare our method (labeled CP in graphs) with the spacing-correlation based scheme proposed by Van Leuken et al. [62] (labeled SC in the graph) and with random vantage points (labeled RND in the graph). All three methods use KD-tree (implemented in C++) for indexing the data in the transformed space.

For each of the methods, we measure the number of distance computations with increasing database size and query radius. Average performance of 100 random queries is used for comparison. As random vantage point method (RND) and spacing correlation based method (SC) do not provide a deterministic set of vantage points, the performance metrics of these methods are not deterministic. Hence, we use average of 10 runs of each experiment to measure the performance. Unless specified otherwise we use 6 vantage points for all the higher dimensional databases and 2 vantage points for 2-D databases. Database sizes were set to 500k for most of the databases except GIS database (80k), fluid dynamics database (200k), weather database (300k) and speech database (50k).

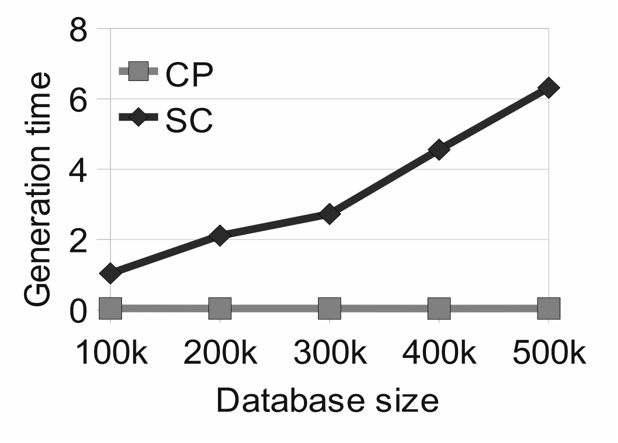
We begin with the comparison of the time required to generate vantage points using our scheme and the one proposed in [62].

6.4.2 Time comparisons for generating vantage points

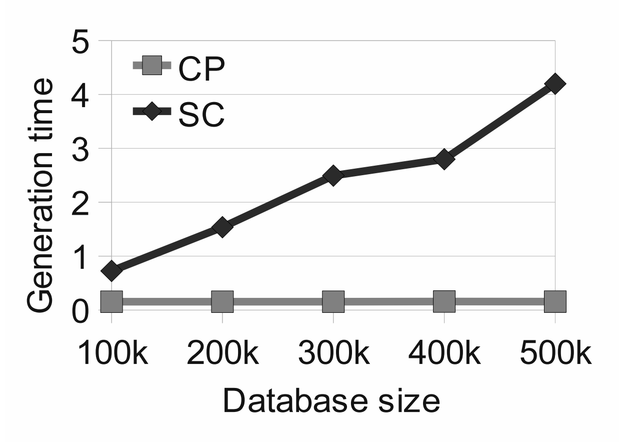
Data dependent method for vantage point generation use various properties of the data to choose the best set of vantage points. Hence, as the database size increases, the time required to generate the vantage points increases. This fact is evident from the graphs in figure 6.12. The figure shows generation times for CP and SC with increasing database size. As the behavior is more or less similar for all the databases, we only show the results for two synthetic databases and two real databases. The vantage point generation time for the



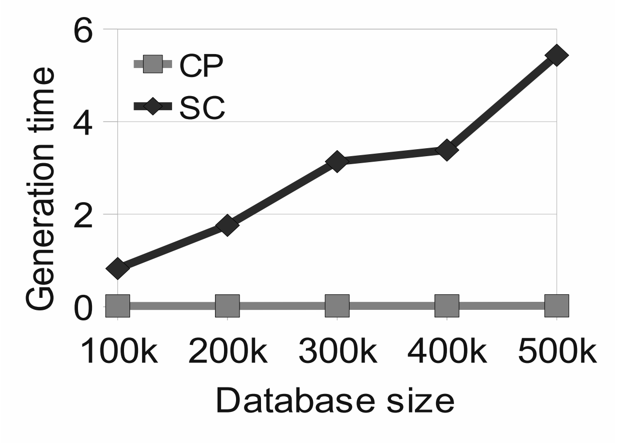
(a) Uniform synthetic data



(b) Clustered synthetic data



(c) Audio feature data



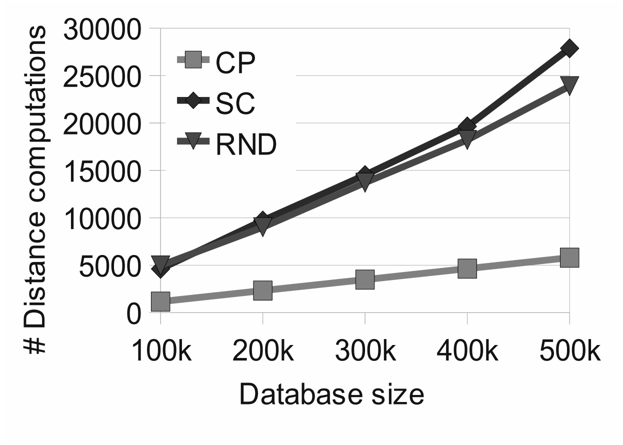
(d) Image feature data

Figure 6.12: Effect of database size on time to generate vantage points

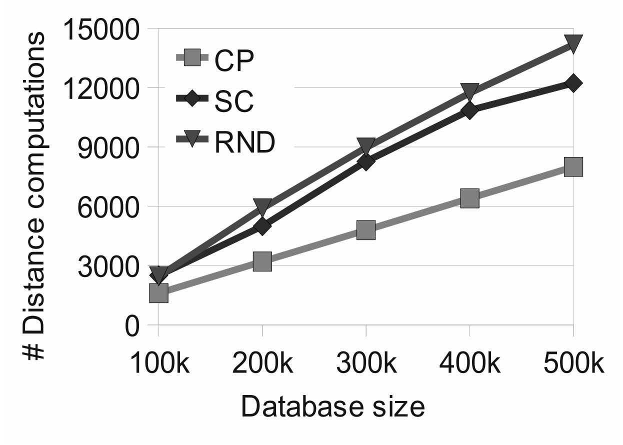
proposed method is few milliseconds in most of the cases.

6.4.3 Range queries in synthetic databases

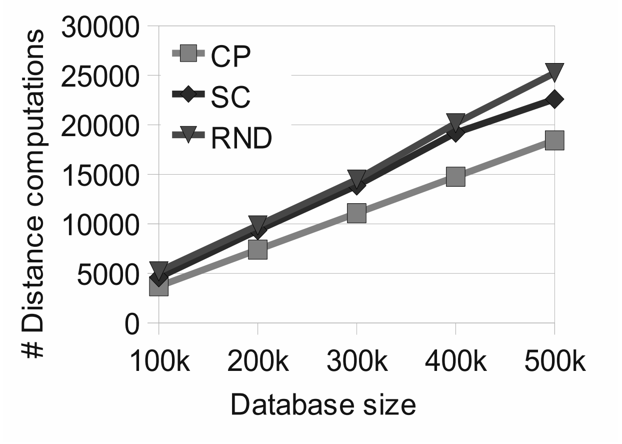
In this section, we compare performance of the three methods (CP, SC and RND) when applied to the different synthetic databases described earlier. We begin with the effect of increasing database size.



(a) Uniform synthetic data



(b) Skewed synthetic data



(c) Clustered synthetic data

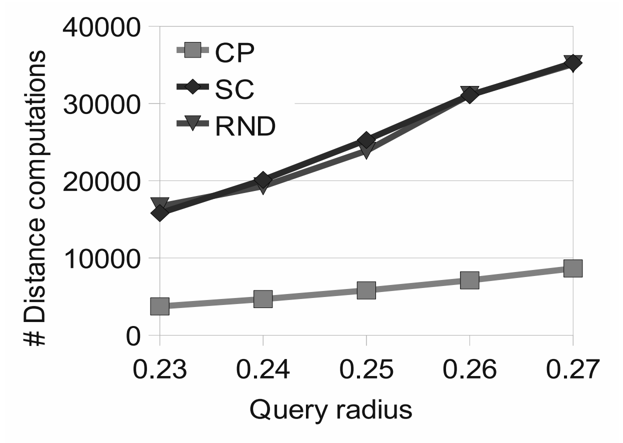
Figure 6.13: Effect of database size on the number of distance computations

6.4.3.1 Effect of database size

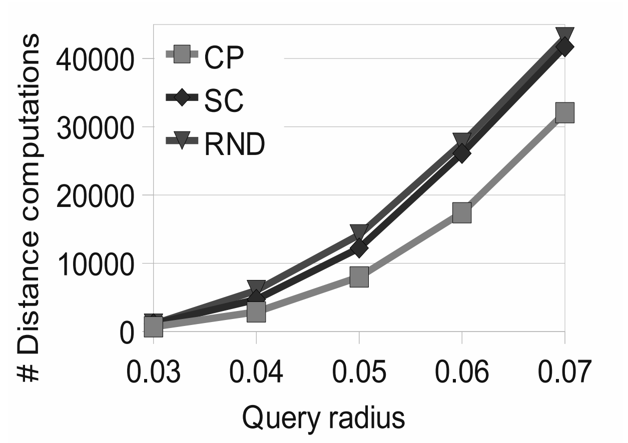
For this set of experiments, database size was increased from 100k to 500k points. Query radius was set to 0.25 for uniform database, 0.05 for skewed data and 0.15 for clustered data. The value of query radius was chosen such that a reasonable number of hits are obtained for each database size. It can be seen from the figure 6.13 that as the database size increases the number of distance computations increases for all the methods. However, as can be seen from the figure, the number of distance computations for CP is consistently less than the other two methods. Further, the rate of increase in the number of distance computations is also lower. This makes corner point method (CP) particularly useful in large databases.

6.4.3.2 Effect of query radius

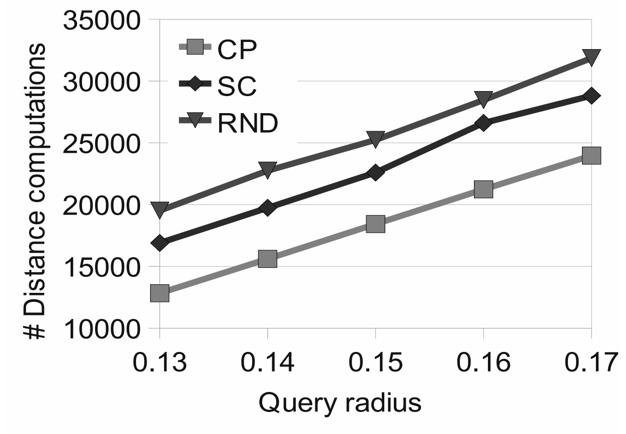
For these experiments, database size was fixed at 500k. The number of vantage points was fixed at 6 as in the previous experiments. As the query radius increases, the query space(volume) increases. In fact, the increase in the query space is exponential in the number of dimensions. If the query is too big then none of the indexing scheme is effective as a



(a) Uniform synthetic data

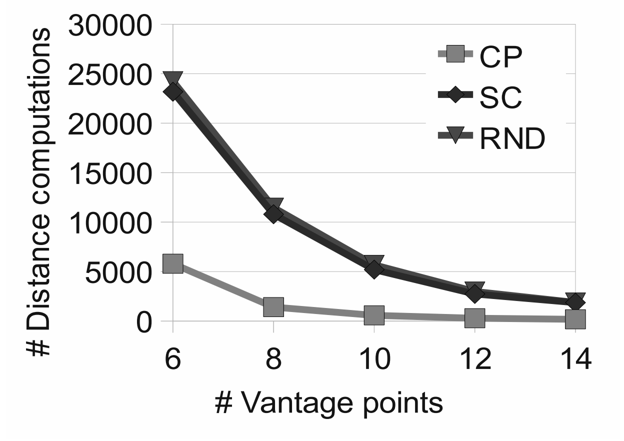


(b) Skewed synthetic data

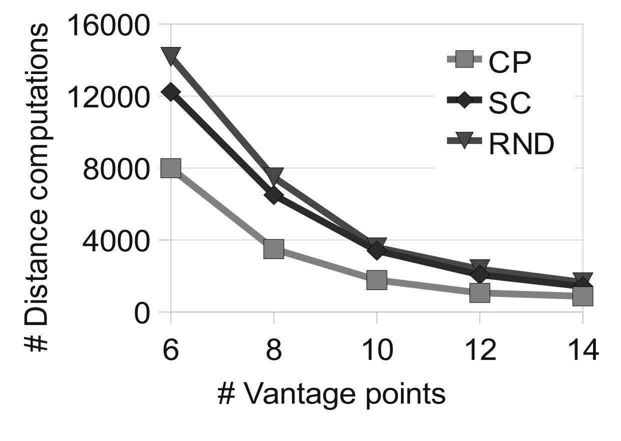


(c) Clustered synthetic data

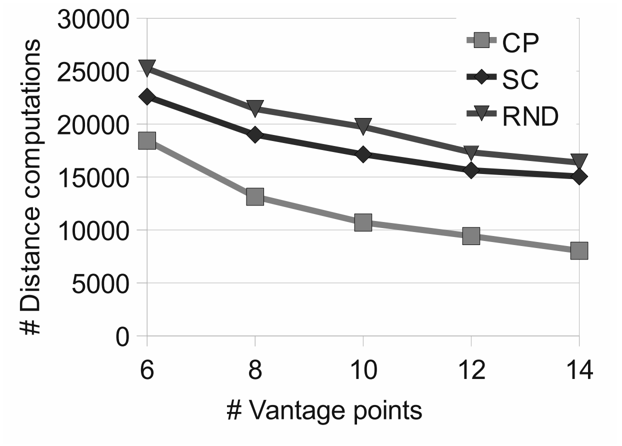
Figure 6.14: Effect of query radius on the number of distance computations



(a) Uniform synthetic data



(b) Skewed synthetic data



(c) Clustered synthetic data

Figure 6.15: Effect of the number of vantage points on the number of distance computations

considerable fraction of the data is retrieved. As shown in the graphs in figure 6.14, with increasing query radius, the proposed data-independent-method, CP, consistently outperforms the other two.

6.4.3.3 Effect of number of vantage points

For these experiments, database size and query radius were fixed. The number of vantage points is increased from 6 to 14 in steps of 2 for each of the methods. Query radii are same as the ones used in section 6.4.3.1. As expected, increasing the number of vantage reduces the number of distance computations. However, the degree of effectiveness of vantage points diminishes with increasing number of vantage points.

6.4.4 Range queries in real databases

We now present result of applying the proposed scheme for real databases. As explained earlier, we use four real databases from different application domains for these experiments. We begin with the performance comparison of the methods with increasing database size.

6.4.4.1 Effect of database size

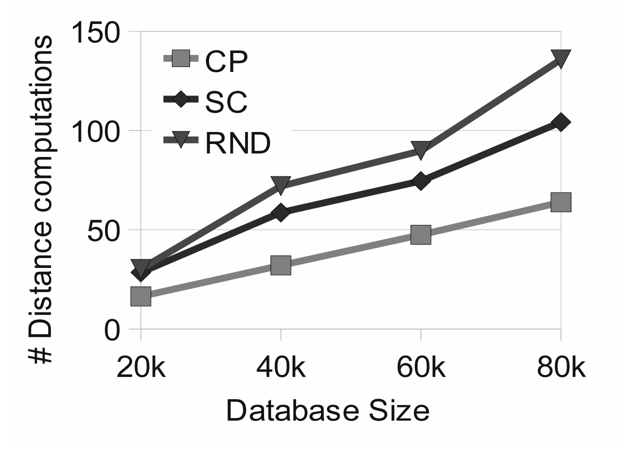
In this set of experiments, query radius and the number of vantage points are kept constant and database size is increased. Query radius was set to 0.05 for audio database, 0.025 for image database, 0.03 for weather database and 0.015 for speech database. Figure 6.16 shows the results of our experiments. It can be seen that the proposed method consistently gives lower number of distance computations than the others.

6.4.4.2 Effect of query radius

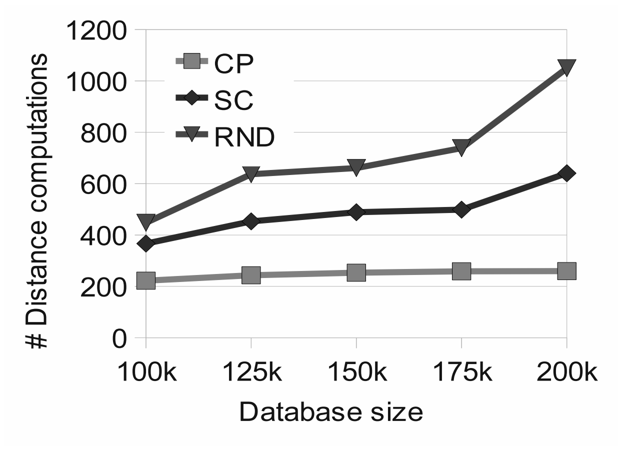
In this set of experiments, we vary the query radius while keeping the database size and the number of vantage points constant. As shown in figure 6.17 with increasing query radius, the number of distance computations increases for all the methods which is quite expected. However, the rate of increase is smaller for the proposed method compared to others.

6.4.4.3 Effect of number of vantage points

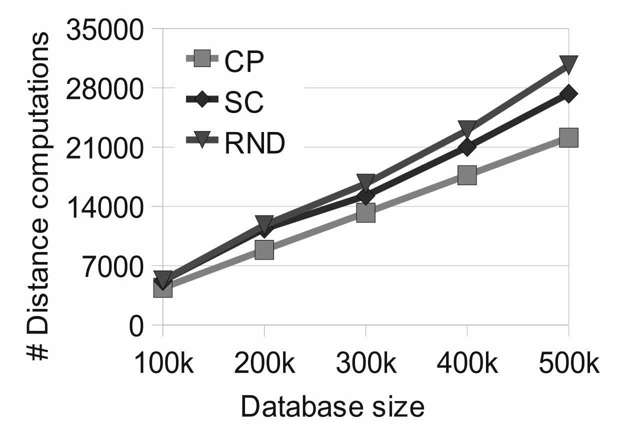
In these experiments, database size and query radius (same as those in 6.4.4.1) were kept constant. The results of increasing the number of vantage points are shown in figure 6.18.



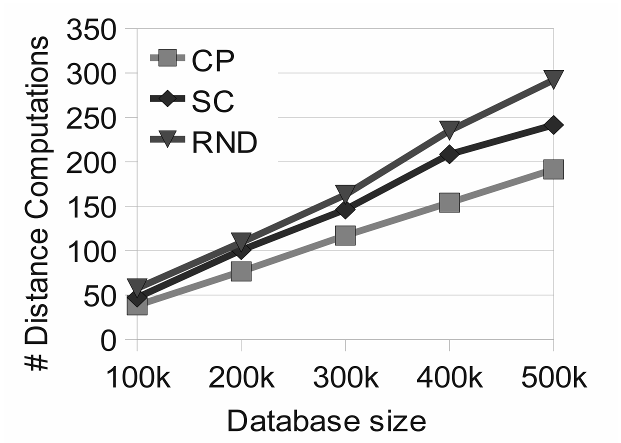
(a) GIS data



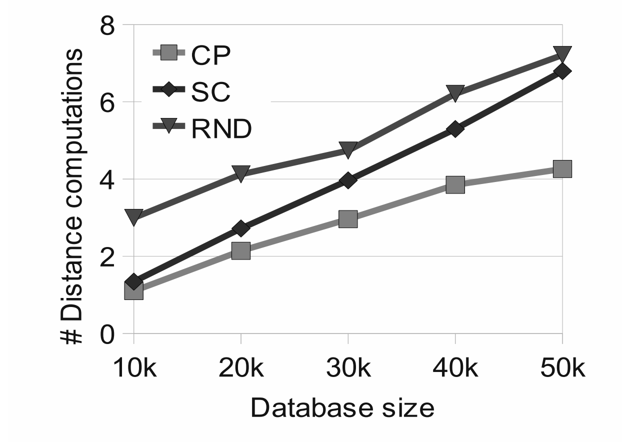
(b) Fluid dynamics data



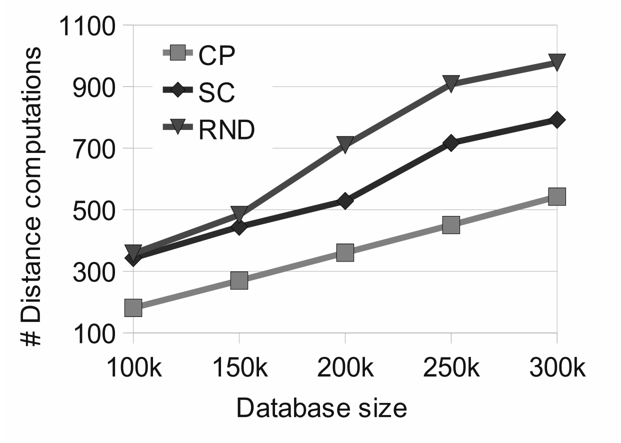
(c) Audio feature data



(d) Image feature data

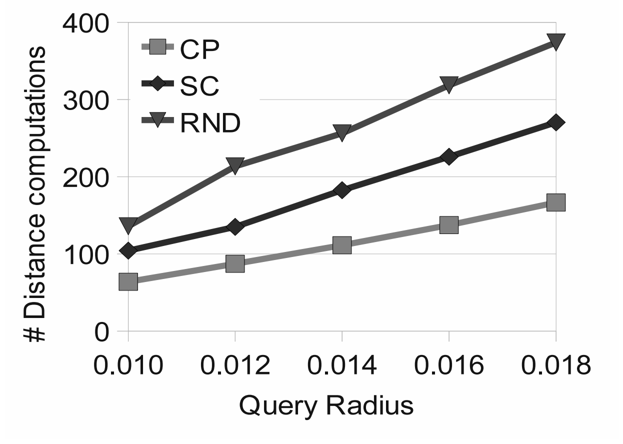


(e) Speech data

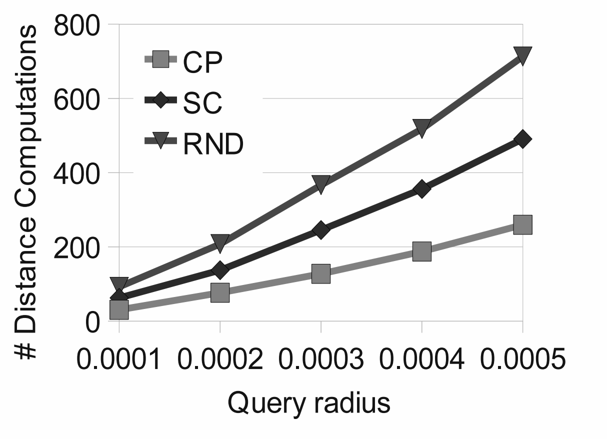


(f) Weather data

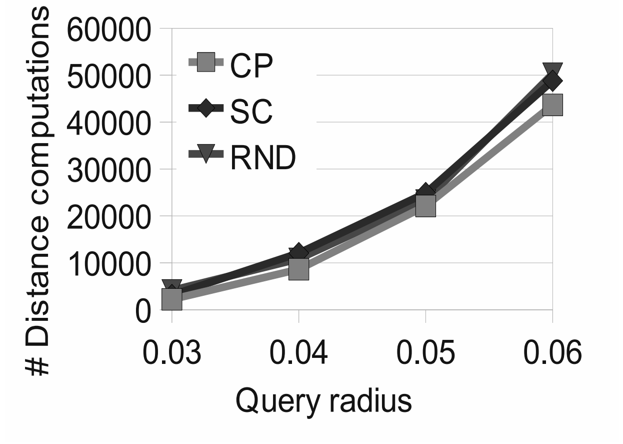
Figure 6.16:]
Effect of database size on the number of distance computations



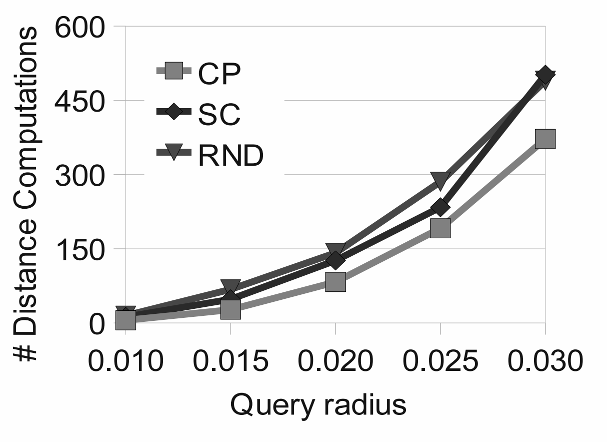
(a) GIS data



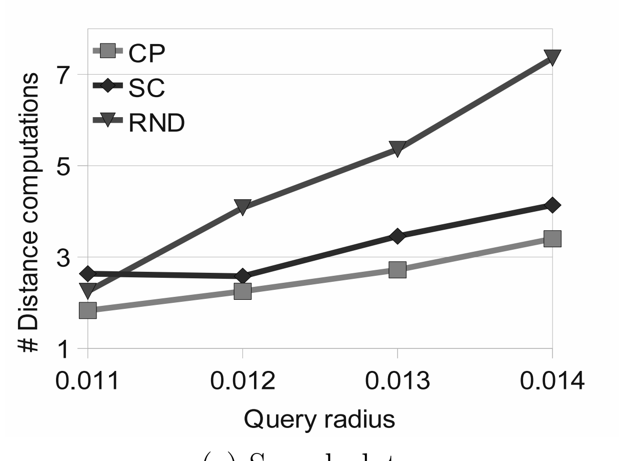
(b) Fluid dynamics data



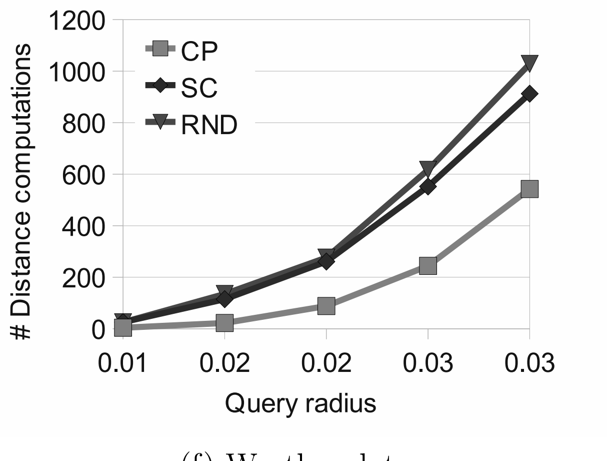
(c) Audio feature data



(d) Image feature data



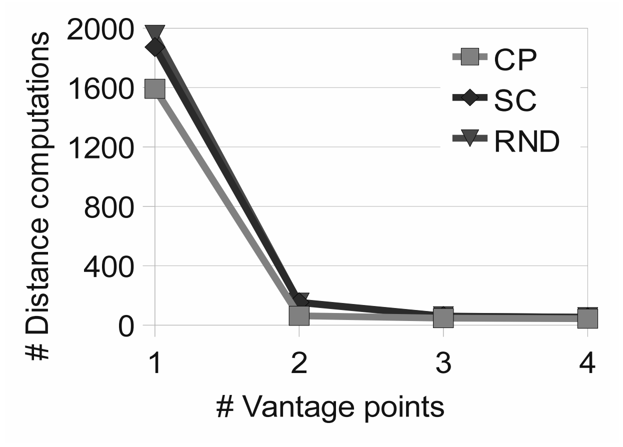
(e) Speech data



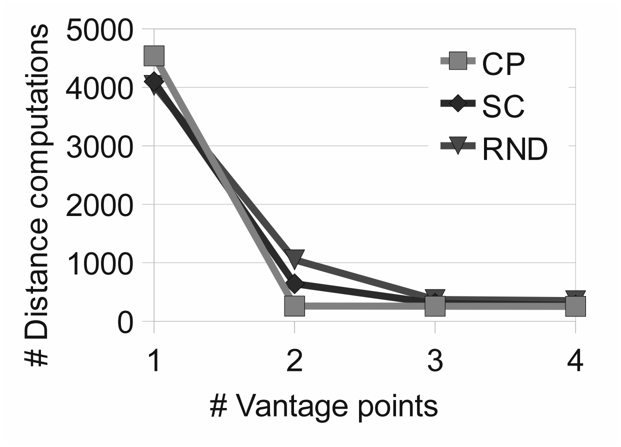
(f) Weather data

Figure 6.17: Effect of query radius on the number of distance computations

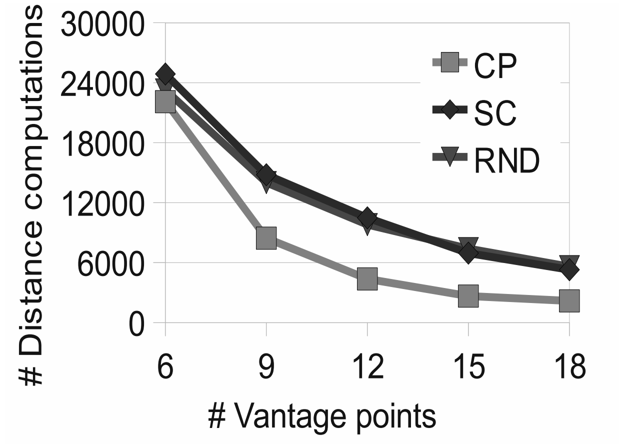
It can be seen that increasing vantage points causes decrease in the number of distance computations and CP method performs the best in all the experiments.



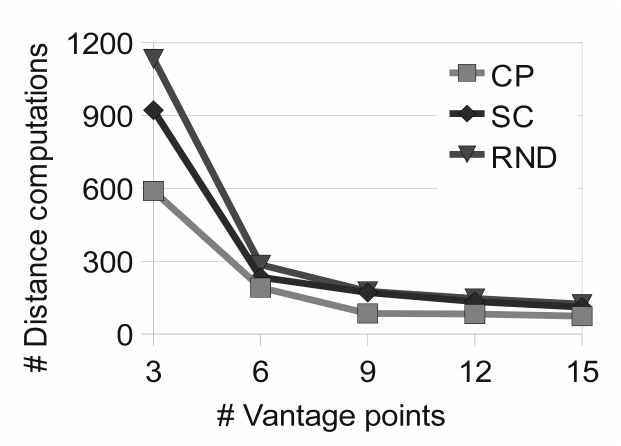
(a) GIS data



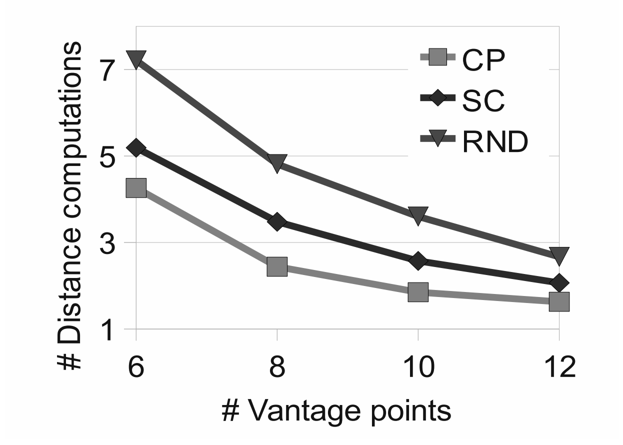
(b) Fluid dynamics data



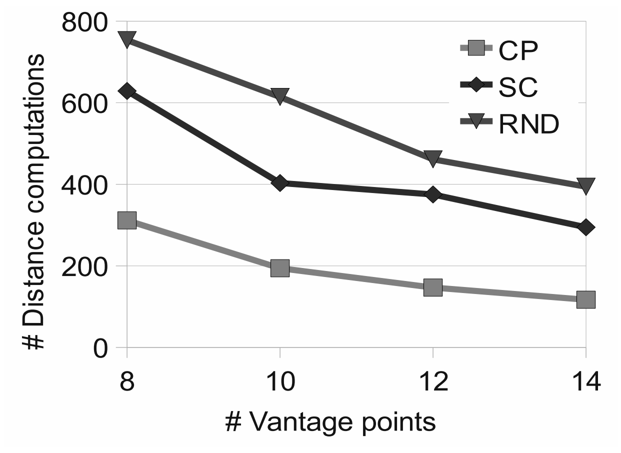
(c) Audio feature data



(d) Image feature data



(e) Speech data



(f) Weather data

Figure 6.18: Effect of the number of vantage points on the number of distance computations

6.5 Vantage point selection in very high dimensional spaces

We observe that there exists certain regularity in the way vantage points are chosen by algorithm 2. This regularity can be exploited to speed up the vantage point generation significantly. In this section, we present a recursive formula for computing a set of vantage points that are exactly at a distance of $\sqrt{\frac{n}{2}}$ from each other. The time complexity of this formula is $O(n \log n)$. However, this formula has two limitations. First, it can only generate vantage points for databases for which the number of dimensions is a power of two. Second, the formula cannot generate more than n points (i.e. number of vantage points cannot be greater than the number of dimensions). The second limitation is not too severe as in most of the applications $m < n$ (since this reduces dimensionality of the data being indexed). To alleviate the first limitations, we present a possible heuristic extension to the formula which can be used when n is not a power of two.

6.5.1 A formula for computing vantage points

When the number of dimensions n is a power of two we can find a set of n vantage points in n dimensional space which are all at a distance of $\sqrt{\frac{n}{2}}$. The formula is recursively defined. Formally, let V_{2k} be the set of $2k$ vantage points in $2k$ dimensional space, and let V_k be the set of k vantage points in k dimensional space. Also, we denote a vantage point $\mathbf{v}$ as a string of 0s and 1s (since they are corner points). Then V_{2k} can be represented recursively as,

$$V_2 = \{00, 01\}$$

$$V_{2k} = \{(\mathbf{v} \cdot \mathbf{v}) | \mathbf{v} \in V_k\} \cup \{(Com(\mathbf{v}) \cdot \mathbf{v}) | \mathbf{v} \in V_k\} \tag{6.5}$$

where, $\mathbf{v} \cdot \mathbf{v}$ represents concatenation of the two strings and $Com(\mathbf{v})$ is a function that complements each character in the string $\mathbf{v}$. It is clear that the formula is applicable only when the number of dimensions is a power of two. We now state and prove correctness of this formula.

Theorem 6.5.1. *Vantage points generated using formula 6.5 are all at a distance of* $\sqrt{\frac{n}{2}}$ *from each other.*

Proof 6.5.1. *Since* n *is a power of 2, we represent it as* $n = 2^k$. *Let* V_{2^k} *be the set of vantage points in* 2^k *dimensional space. We will prove the theorem using mathematical induction on* k. *The basis case corresponds to* $k = 1$. *It can be easily seen that the vantage points in 2-D space are indeed at a distance of* $\sqrt{2/2} = 1$ *from each other.*

Let us assume that the theorem is true for some k. *Consider* $V_{2^{k+1}}$. *From the formula 6.5, every point in this set is generated either as a repetition of a point in* V_{2^k} *or as a combination of a point and its complement in* V_{2^k}. *Let 4 points in* $V_{2^{k+1}}$ *be,* $\mathbf{v_1} = (u \cdot u)$, $\mathbf{v_2} = (u \cdot Com(u))$, $\mathbf{v_3} = (v \cdot v)$, $\mathbf{v_4} = (v \cdot Com(v))$, *where* $u, v \in V_{2^k}$. *From induction hypothesis,* u *and* v *differ in exactly* $\frac{2^k}{2}$ *bits. The first halves of* $\mathbf{v_1}$ *and* $\mathbf{v_2}$ *are same while the second halves are complements of each other hence, these two points differ in* 2^k *bits which means their distance is* $\sqrt{2^k} = \sqrt{\frac{2^{k+1}}{2}}$. *Same argument applies for* $\mathbf{v_3}$ *and* $\mathbf{v_4}$. *Both halves of* $\mathbf{v_1}$ *and* $\mathbf{v_3}$ *differ in* $\frac{2^k}{2}$ *bits resulting in their distance being* $\sqrt{\frac{2^{k+1}}{2}}$. *Note that if* u *and* v *differ* *in $\frac{2^k}{2}$ bits then $Com(u)$ and v also differ in $2^k - \frac{2^k}{2} = \frac{2^k}{2}$ bits. In fact, extending the same argument, we can see that, $Com(u)$ and $Com(v)$ will also differ in $\frac{2^k}{2}$ bits. Once these facts are taken into consideration, it can be easily seen that the points in other three pairs, i.e. $\{\mathbf{v_1}, \mathbf{v_4}\}$, $\{\mathbf{v_2}, \mathbf{v_3}\}$ and $\{\mathbf{v_2}, \mathbf{v_4}\}$ are also at a distance of $\sqrt{\frac{2^{k+1}}{2}}$ from each other. Since, every point in $V_{2^{k+1}}$ is generated from some point $u \in V_{2^k}$, it can be argued that all the points in $V_{2^{k+1}}$ are at a distance of $\sqrt{\frac{2^{k+1}}{2}}$ from each other.*

Following is an example showing the generation of the sets V_2, V_4 and V_8.

Example 6.5.1. *Vantage points for 2-D, 4-D and 8-D spaces*

V_2	V_4	V_8
0 0	0 0 0 0	0 0 0 0 0 0 0 0
0 1	1 1 0 0	1 1 1 1 0 0 0 0
	0 1 0 1	1 1 0 0 1 1 0 0
	1 0 0 1	0 0 1 1 1 1 0 0
		0 1 0 1 0 1 0 1
		1 0 1 0 0 1 0 1
		1 0 0 1 1 0 0 1
		0 1 1 0 1 0 0 1

The requirement that the number of dimensions must be a power of 2 may be very restrictive. We now present an algorithm to generate vantage points for any number of dimensions by extending the formula in equation 6.5. The main idea is to start with the next higher power of two and then eliminate dimensions one by one to get to the desired dimensionality.

Algorithm 3 highlights the main steps in generating vantage points for any number of dimensions. It begins with the vantage points for the next higher dimension which is a power of 2 (i.e., for $n = 12$, it will begin with vantage points in 16 dimensions). Then for each dimension, it calculates the pairwise distance variance of points obtained after that dimension is removed. Dimension whose removal results in the minimum variance, is

Algorithm 3 Generate m vantage points for n dimensional unit hypercube

Input: Number of dimensions n and number of vantage points m.

Output: A set of m vantage points.

Algorithm

```
1: Let $n'$ be the power of 2 just larger than $n$
2: Generate the initial set of vantage points in $n'$ dimensions. $V_{n'}$ using formula in equation
   6.5
3: Initialize $V \leftarrow V_{n'}$
4: for $i = 1$ to $n' - n$ do
5:     minDistVariance = $\infty$
6:     for each dimension $c$ not yet removed do
7:         $V_{temp}$ = Set of unique vantage points obtained after elimination of dimension $c$ from
           $V$
8:         distVariance = GetDistanceVariance($V_{temp}$)
9:         if distVariance < minDistVariance then
10:            minDistVariance = distVariance
11:            $next = c$
12:        end if
13:    end for
14:    $V$ = Set unique vantage points obtained after elimination of dimension $next$ from $V$
15: end for
16: return $V$
```

removed in each iteration. The process is repeated until desired number of dimensions are removed. Note that, this algorithm does not guarantee that the resulting vantage points will have the minimum possible pairwise distance variance. However, our experiments suggest that minimum pairwise distance variance of vantage point obtained using this algorithm is close to those obtained using algorithm 2. What is very important here is that algorithm 3 has polynomial time complexity compared to exponential complexity of algorithm 2 which makes it a more attractive choice in high dimensional spaces.

6.5.2 Selection vantage points based on number of flips

The formula presented in previous section generates n vantage points. However, in most of the practical cases, the number of vantage points m is considerably less n. In theory,

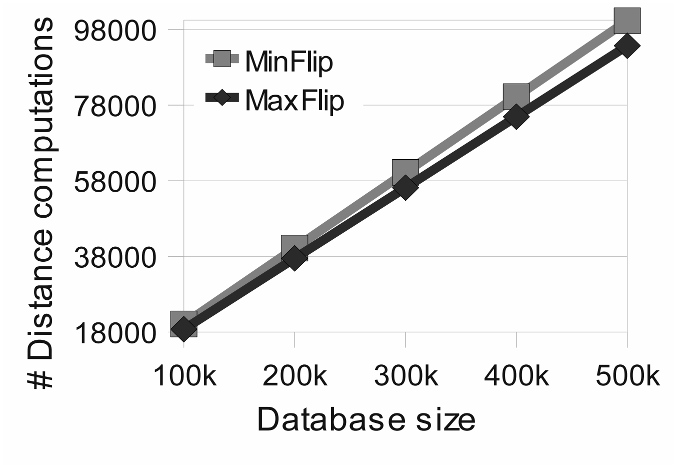
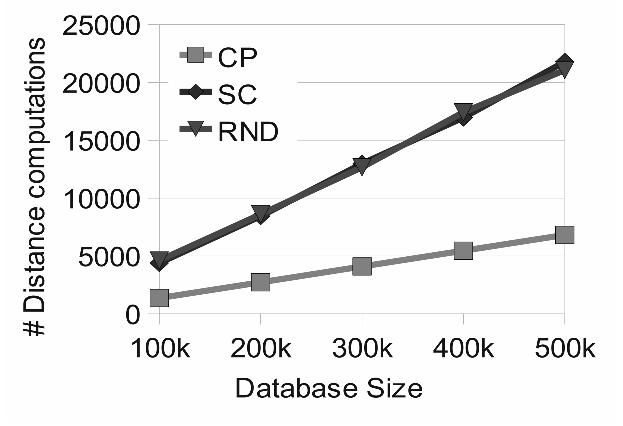
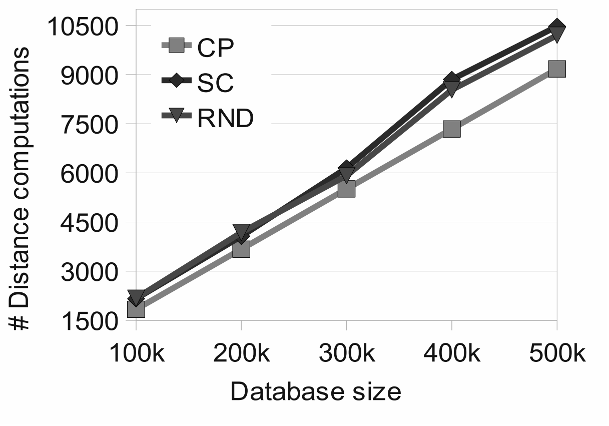


Figure 6.19: Performance comparison of vantage points selection with minimum and maximum flips

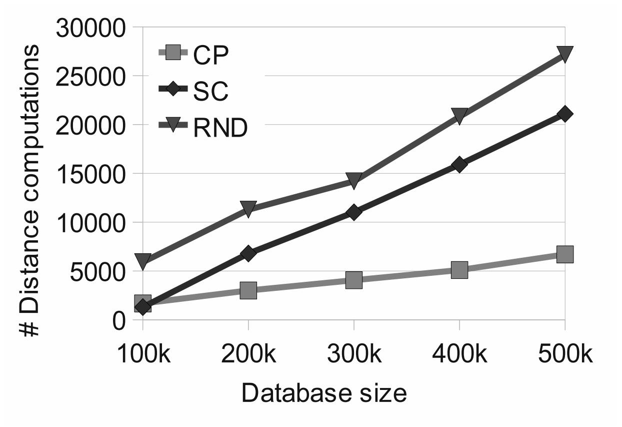
any set of m points out of these n candidates will be equally good (in terms of pairwise distance variance). However, we observed that there is difference in the performance of the range query depending on which particular set of vantage points is chosen. We define a flip as change in value of dimension i with respect to dimension $i-1$. For example, in point $(0,1,1,0)$, there is a flip in second and fourth dimension. It can be easily seen that the points obtained using formula 6.5, have 1-point each with 0 to $n-1$ flips. Our experiments with synthetic data suggest that selecting points with most number of flips tends to have consistently better (about 7 to 10%) query performance than those with least number of flips. Figure 6.19 shows the comparison of number of distance computations obtained when using vantage points with maximum number of flips and those with minimum number of flips. These experiments are run with 16-dimensional uniform synthetic data. It can be clearly seen that using vantage point with more flips may be slightly more beneficial. Assuming $n = 4$ and $m = 2$, this heuristic chooses the following set of vantage points : $(0,1,0,1), (1,0,0,1)$.



(a) Uniform synthetic data



(b) Skewed synthetic data

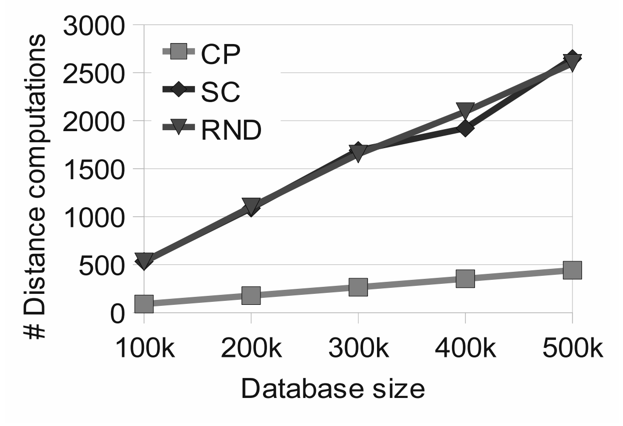


(c) Clustered synthetic data

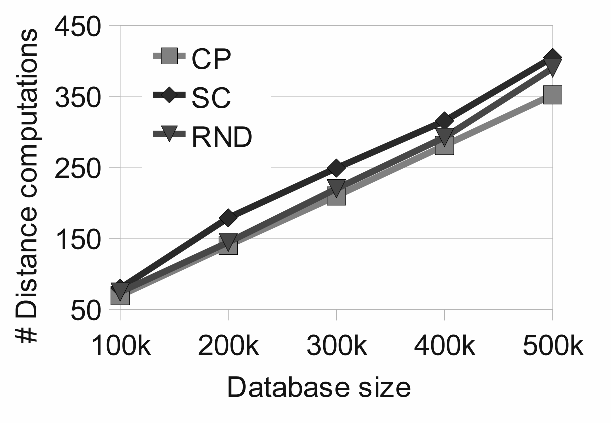
Figure 6.20: Applying the proposed scheme on 64 dimensional synthetic data

6.5.3 Performance comparison in very high dimensional space

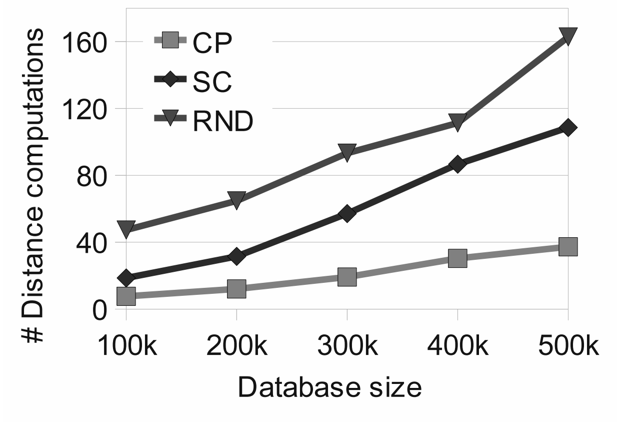
Using the formula from section 6.5.1, we computed 8 vantage points for 64 and 60 dimensional synthetic data. Vantage points for 64 dimensions can be directly generated using equation 6.5, while for algorithm 3 for 60 dimensions. Query range was set to values between 0.05 to 0.3 for each of the experiments. We compared the performance of the three methods, CP, SC and RND, for uniform, clustered and skewed data sets. The performance comparison, shown in figures 6.20 and 6.21 clearly demonstrates the superiority of the corner point approach.



(a) Uniform synthetic data



(b) Skewed synthetic data



(c) Clustered synthetic data

Figure 6.21: Applying the proposed scheme on 60 dimensional synthetic data

6.6 *k*-NN queries in vantage space

k-NN queries form an important class of queries in high dimensional databases. Although *k*-NN queries are similar to range queries from the implementation point of view, there exists an important difference between the two. The order of retrieval of query results does not matter for range queries. For *k*-NN queries on the hand, the order is very important. A vantage point based transformation does not preserve distances or the ordering among the data points. Hence, implementing *k*-NN queries using the transformation can be challenging. In this section, we present preliminary results on implementation of *k*-NN queries using the

Table 6.6: Recall of k-NN queries on uniform 2-D data (database size = 100k)

# of neighbors (k)	**# Vantage points**		
	2	**3**	**4**
1	0.79	0.9	0.94
25	0.82	0.88	0.94
50	0.83	0.88	0.94
75	0.83	0.88	0.94
100	0.83	0.88	0.94

Table 6.7: Recall of k-NN queries on uniform 10-D data (database size = 100k)

# of neighbors (k)	**# Vantage points**			
	6	**8**	**10**	**12**
1	0.03	0.08	0.27	0.38
25	0.12	0.27	0.44	0.56
50	0.14	0.30	0.45	0.56
75	0.16	0.31	0.46	0.57
100	0.17	0.33	0.47	0.58

vantage point transformation.

Tables 6.6 and 6.7 summarize the results of running the k-NN query in the transformed space for increasing values of k. In each of the experiments, correctness of the k-NN query in the transformed space is measured using recall as the quality metric. If $\mathbf{D_k}$ is the correct set of k neighbors and if $\mathbf{D'_k}$ is the set of k points retrieved by the query, then Recall R of the query is defined as,

$$R = \frac{|\mathbf{D'_k} \cap \mathbf{D_k}|}{|\mathbf{D_k}|}$$

It can be seen from these two tables that, at higher dimensions, recall of the k-NN queries is low. However, for a given number of dimensions, as k increases, recall gets better. Increasing the number of vantage points also improves the recall significantly. We also observe that k-NN queries run in the transformed space tend to have lower number of distance computations. For example, for $k = 10$, query in the original space has about 18280 distance computations while the same query in vantage space (with 8 vantage points) has only 5672 distance computations. The number of distance computations increases with the increasing number of vantage points. At first, this may seem rather counter intuitive. But it should be noted that by increasing the number of vantage points, we are increasing the dimensionality of the transformed space. KD-tree like many other indexing schemes suffers from dimensionality curse and hence provides poor performance in higher dimensions.

From these preliminary results, it can be concluded that there is a promise in the transformation based approaches for k-NN queries, when the number of vantage points is less than the number of dimensions. However, a naive k-NN query algorithm may not be sufficient to provide a good recall . By using more advanced heuristics, it may be possible to improve the recall statistics of the k-NN queries.

Chapter 7

Conclusion and Future Work

In this dissertation, we present several novel techniques that enable mapping of a range query onto a box query and vice versa. It can be seen that under certain conditions, range queries and box queries are not too different and one type of query can be implemented as another enabling us to take advantage of specific properties of an index.

Our first transformation, maps L_1 range queries onto box queries. The proposed transformation is computationally easy to implement. In an R-tree like index, this mapping provides similar interface of query spaces with space of the index pages. The idea of disjoint planar rotations coupled with pruning box query can be used to achieve this mapping even in higher dimensional spaces without any false positives. The theoretical analysis shows that this mapping can provide improvement in query execution performance. The improvement is dependent on the relative sizes of the query ranges and box sizes; and increases with increasing dimensions, which is verified by experiments with synthetic as well as real data. As k-NN queries are similar to range queries, the transformation can be applied to improve the performance of k-NN queries as well. We show that the dynamic index structures like the

R*-tree can provide improved performance using this transformation but suffer from large variance in results. However, using static indexing schemes such as the packed R-tree, the performance improvement is consistent and in agreement with our theoretical model.

The second transformation maps a box query to a L_1 range query. Although the basic concept and the premise of the mapping is similar, the fact that a box query can have non-uniform box sizes poses some challenges in the transformation. We propose the concept of multiple queries to handle these and show that the resulting range query can significantly outperform the original box query in M-tree like index structures.

We explore several possible transformations of L_2 range queries onto box queries. Due to specific properties of the L_2 space, it may not be possible to precisely map L_2 range queries onto box queries without any false positives. However, L_2 range queries can be mapped to minimum bounding box queries using vantage point based schemes for main memory database applications. Previous work in this area shows that this approach can provide significant reduction in the number of distance computations required to execute the query. Selection of vantage points is a critical step in this process and most of the existing schemes for vantage point selection are data dependent. As a result, they do not support dynamic insertion, deletion and updates in the data. Further, their running time for selection the vantage points is proportional to the database size. We present a thorough analysis of the impact of selection of vantage points on the number of false positives. Based on these analyses, we present a novel data-independent scheme for selection of vantage points in closed data spaces. Being data independent, our scheme allows dynamic insertions, deletions and updates. Comprehensive experimental evaluation with several synthetic and real databases shows that our technique for selection of vantage points provides much better performance from the existing methods. The time complexity of our vantage point selection algorithm is exponential in number of dimensions which may be problematic if the number of dimensions is very large. We propose a recursive mathematical formula which generates the ideal set of vantage points when the number of dimensions is a power of two and has a time complexity $O(n \log n)$. We also propose an extension to the formula which can be used for any number of dimensions and runs in polynomial time. We show that these methods, when applied to very high dimensional data, provide significant performance improvement over existing techniques.

In high dimensional data spaces, k-NN queries in main memory databases are equally important. The future work involves evaluation of the applicability of the proposed vantage point selection scheme for k-NN queries in high dimensional spaces. As the vantage point based transformation is not order preserving, the nearest neighbors in the transformed space may not be real nearest neighbors in the original space. Increasing the number of vantage points may improve the quality of results, but, since the KD-tree is known to suffer from dimensionality curse, an increase in the number of vantage points can significantly affect search performance of the KD-tree. This may require a novel algorithm for implementing k-NN search such that good quality results can be obtained without sacrificing performance. Although vantage point concepts have been used for disk based database systems, they too use data dependent vantage points. Hence, application of the proposed data independent vantage selection techniques in these systems needs to be studied further.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] National oceanic and atmospheric administration (NOAA) weather data - `ftp://ftp.ncdc.noaa.gov/pub/data/gsod/`.

[2] C. C. Aggarwal. On the effects of dimensionality reduction on high dimensional similarity search. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 256–266, New York, NY, USA, 2001. ACM.

[3] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *Proceedings of the 8th International Conference on Database Theory*, ICDT '01, pages 420–434, London, UK, 2001. Springer-Verlag.

[4] G. Amato and P. Savino. Approximate similarity search in metric spaces using inverted files. In *Proceedings of the 3rd international conference on Scalable information systems*, InfoScale '08, pages 28:1–28:10, ICST, Brussels, Belgium, Belgium, 2008.

[5] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, SODA '93, pages 271–280, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[6] R. Bayer. The universal B-tree for multidimensional indexing: general concepts. In *Proceedings of the International Conference on Worldwide Computing and Its Applications*, WWCA '97, pages 198–209, London, UK, 1997. Springer-Verlag.

[7] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: an efficient and robust access method for points and rectangles. *SIGMOD Rec.*, 19(2):322–331, 1990.

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

[9] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, September 1975.

[10] T. Bertin-Mahieux, D. P. E. nad Brian Whitman, and P. Lamere. The million song dataset. 2011.

[11] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory*, ICDT '99, pages 217–235, London, UK, UK, 1999. Springer-Verlag.

[12] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proceedings of the 1997 ACM SIGMOD international conference on Management of data*, SIGMOD '97, pages 357–368, New York, NY, USA, 1997. ACM.

[13] T. Bozkaya and M. Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Trans. Database Syst.*, 24:361–404, September 1999.

[14] S. Brin. Near neighbor search in large metric spaces. In *Proceedings of the 21th International Conference on Very Large Data Bases*, VLDB '95, pages 574–584, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[15] N. R. Brisaboa, A. Farina, O. Pedreira, and N. Reyes. Similarity search using sparse pivots for efficient multimedia information retrieval. In *Proceedings of the Eighth IEEE International Symposium on Multimedia*, ISM '06, pages 881–888, Washington, DC, USA, 2006. IEEE Computer Society.

[16] B. Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recogn. Lett.*, 24:2357–2366, October 2003.

[17] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Trans. Database Syst.*, 27(2):188–228, 2002.

[18] E. Chavez Gonzalez, K. Figueroa, and G. Navarro. Effective proximity retrieval by ordering permutations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30:1647–1658, September 2008.

[19] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 426–435, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[20] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[21] C. Faloutsos and K.-I. Lin. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, SIGMOD '95, pages 163–174, New York, NY, USA, 1995. ACM.

[22] R. Fenk, V. Markl, and R. Bayer. Interval processing with the ub-tree. In *Proceedings of the 2002 International Symposium on Database Engineering & Applications*, IDEAS '02, pages 12–22, Washington, DC, USA, 2002. IEEE Computer Society.

[23] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3:209–226, September 1977.

[24] G. W. Furnas, S. Deerwester, S. T. Dumais, T. K. Landauer, R. A. Harshman, L. A. Streeter, and K. E. Lochbaum. Information retrieval using a singular value decomposition model of latent semantic structure. In *Proceedings of the 11th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '88, pages 465–480, New York, NY, USA, 1988. ACM.

[25] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30:170–231, June 1998.

[26] C. Gennaro, G. Amato, P. Bolettieri, and P. Savino. An approach to content-based image retrieval based on the lucene search engine library. In *Proceedings of the 14th European conference on Research and Advanced Technology for Digital Libraries*, ECDL'10, pages 55–66, Berlin, Heidelberg, 2010. Springer-Verlag.

[27] J. E. Gentle. *Numerical Linear Algebra for Applications in Statistics.* Springer-Verlag, 1998.

[28] M. Gromov. Finite propagation speed, kernel estimates for functions of the laplace operator, and the geometry of complete riemannian manifolds.

[29] M. Gromov. Groups of polynomial growth and expanding maps. pages 53–73, 1981.

[30] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, SIGMOD '84, pages 47–57, New York, NY, USA, 1984. ACM.

[31] C. Hennig and L. J. Latecki. The choice of vantage objects for image retrieval. *Pattern Recognition*, 36:2187–2196, October 2003.

[32] D. Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück . *Mathematische Annalen*, pages 459–460, 1890.

[33] K. Hinrichs. Implementation of the grid file: design concepts and experience. *BIT*, 25(4):569–592, Dec. 1985.

[34] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Trans. Database Syst.*, 24:265–318, June 1999.

[35] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces (survey article). *ACM Trans. Database Syst.*, 28(4):517–580, 2003.

[36] D. Hull. Improving text retrieval for the routing problem using latent semantic indexing. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '94, pages 282–291, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[37] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):1–58, 2008.

[38] K. ip Lin, H. V. Jagadish, and C. Faloutsos. The tv-tree: An index structure for high-dimensional data. *The Vldb Journal*, 3:517–542, 1994.

[39] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. idistance: An adaptive b+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.*, 30:364–397, June 2005.

[40] I. T. Jolliffe. *Principal Component Analysis (Springer series in statistics)*. Springer, 2002.

[41] I. Kamel and C. Faloutsos. On packing R-Trees. In *Proceedings of the second international conference on Information and knowledge management*, CIKM '93, pages 490–499, New York, NY, USA, 1993. ACM.

[42] R. Kenyon. Tiling a rectangle with the fewest squares. *Journal of Combinatorial Theory*, A 76:272–291, 1996.

[43] E. J. Keogh, K. Chakrabarti, M. J. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowl. Inf. Syst.*, 3:263–286, 2001.

[44] H.-P. Kriegel and B. Seeger. Plop-hashing: A grid file without directory. In *Proceedings of Fourth International Conference on Data Engineering*, pages 369 –376, Feb. 1988.

[45] S. Lang. *Linear Algebra*. New York: Springer-Verlag, 1987.

[46] S. T. Leutenegger. Multi dimensional data sets, 2012.

[47] R. Mao, W. L. Miranker, and D. P. Miranker. Dimension reduction for distance-based indexing. In *Proceedings of the Third International Conference on SImilarity Search and APplications*, SISAP '10, pages 25–32, New York, NY, USA, 2010. ACM.

[48] D. J. Mavriplis. An advancing front delaunay triangulation algorithm designed for robustness. *J. Comput. Phys.*, 117(1):90–101, Mar. 1995.

[49] MIT image dataset. MIT CSAIL : Visual dictionary - `http://groups.csail.mit.edu/vision/TinyImages/`, 2010.

[50] G. M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. *Technical report*, 1966.

[51] A. H. H. Ngu, Q. Z. Sheng, D. Q. Huynh, and R. Lei. Combining multi-visual features for efficient indexing in a large image database. *The VLDB Journal*, 9:279–293, April 2001.

[52] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. H. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin. The QBIC project: Querying images by content, using color, texture, and shape. In *Storage and Retrieval for Image and Video Databases*, pages 173–187, 1993.

[53] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst.*, 9:38–71, March 1984.

[54] J. Orenstein. A comparison of spatial query processing techniques for native and parameter spaces. *SIGMOD Rec.*, 19:343–352, May 1990.

[55] S. Pramanik, A. Watve, C. R. Meiners, and A. Liu. Transforming range queries to equivalent box queries to optimize page access. *Proc. VLDB Endow.*, 3:409–416, September 2010.

[56] K. V. Ravi Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. *SIGMOD Rec.*, 27(2):166–176, 1998.

[57] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, SIGMOD '95, pages 71–79, New York, NY, USA, 1995. ACM.

[58] H. Tropf and H. Herzog. Multidimensional range search in dynamically balanced trees. *Applied Informatics*, pages 71–77, 1981.

[59] UCIML Repository. UCI machine learning repository corel image feature data set - `http://archive.ics.uci.edu/ml/datasets/Corel+Image+Features`, 2010.

[60] J. K. Uhlmann. Metric trees. *Applied Mathematics Letters*, 4(5):61 – 62, 1991.

[61] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Inf. Process. Lett.*, 40(4):175–179, 1991.

[62] R. H. Van Leuken and R. C. Veltkamp. Selecting vantage objects for similarity indexing. *ACM Trans. Multimedia Comput. Commun. Appl.*, 7:16:1–16:18, September 2011.

[63] J. Venkateswaran, D. Lachwani, T. Kahveci, and C. Jermaine. Reference-based indexing of sequence databases. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 906–917. VLDB Endowment, 2006.

[64] K. Vu, K. A. Hua, H. Cheng, and S.-D. Lang. A non-linear dimensionality-reduction technique for fast similarity search in large databases. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 527–538, New York, NY, USA, 2006. ACM.

[65] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[66] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the fourth annual ACM-SIAM Symposium on Dis-*

crete algorithms, SODA '93, pages 311–321, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.