A NEW DYNAMIC AND ADAPTIVE SCHEME FOR INDEXING IN METRIC SPACES

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Umut TOSUN

August, 2007

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Dr. Cengiz Çelik (Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Özgür Ulusoy (Co-Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Çiğdem Gündüz Demir

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Pınar Duygulu Şahin

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Dr. Tansel Özyer

Approved for the Institute of Engineering and Science:

_____

Prof. Dr. Mehmet B. Baray
Director of the Institute of Engineering and Science

ABSTRACT

A NEW DYNAMIC AND ADAPTIVE SCHEME FOR INDEXING IN METRIC SPACES

Umut TOSUN
M.S. in Computer Engineering
Supervisor: Dr. Cengiz Çelik
Co-Supervisor: Prof. Dr. Özgür Ulusoy
August, 2007

Computer Science applications are often concerned with efficient storage and retrieval of data. Well defined structure of traditional databases help to access required query objects effectively using the *Relational Database* paradigm. However, in recent times, we are faced with the challenges of dealing with unstructured and complex data such as images, video, sound clips and text documents. Multimedia Information Retrieval, Data Mining, Pattern Recognition, Machine Learning, Computer Vision and Biomedical Databases are examples of the fields that require efficient management of complex data. Complex, unstructured type of data often cannot be broken down into well-defined components, and exact matching cannot be applied for defining queries. Instead, the notion of *similarity search* is used where a query or prototype object is provided by the user and the database retrieves the objects that are similar.

One popular approach for similarity searching is to approximate the relationship between database objects by mapping them into a vector space. There are well-known indexing methods in literature that support similarity queries in vector spaces, however, it has been shown that these methods are ineffective for high dimensional data. Another approach is to use *Metric Spaces* model for indexing. Metric spaces are defined by a distance function that has the triangular inequality property. Since there are no assumptions about the structure of the data itself, they constitute a higher level abstraction and thus have more applicability. They have also been shown to perform better in higher dimensions.

A lot of the previous work in metric spaces have concentrated on static methods that do not allow new insertions once the index structure has been initialized.

M-Tree, Slim-Tree, DF-Tree, Omni are some of the popular dynamic structures. These methods can grow incrementally by splitting overflowed nodes and adding new levels to the tree very much like the B-tree variants. Unfortunately, they have been shown to perform very poorly compared to flat structures such as AESA, LAESA, Spaghettis and Kvp that use a fixed set of global pivots. The distances between the query object and the pivots are computed to eliminate some portion of the database from consideration. The number of pivots can be easily increased to provide more selectivity, thus better query performance. However, there is an optimum number of pivots for a given query radius, and using too many pivots increases the costs of queries and the initialization of the index. Recently, Sparse Spatial Selection(SSS) was introduced as a LAESA variant that allows insertions of new database objects and dynamically promotes some of the new objects as pivots.

In this thesis, we argue that SSS has fundamental problems that results in poor query performance for clustered or otherwise skewed distributions. Real datasets have often been observed to show such characteristics. We show that SSS has been optimized to work for a symmetrical, balanced distribution and for a specific radius value. Our first main contribution is offering a new pivot promotion scheme that can perform robustly for clustered or skewed distributions. Our second contribution is proposing new methods that solve the problem of determining the right number of pivots for different query radius values. We show that our new indexing scheme performs significantly better than tree-based dynamic structures while having lower insertion costs. We also show that our structure adapts to changes in the database population in a superior way.

Keywords: Metric Space, Metric Access Methods, Kvp, Hkvp, EcKvp, M-Tree, Slim-Tree, DF-Tree, Pivot, Distance Computation.

ÖZET

METRİK UZAYLARDA İNDEKSLEME İÇİN DİNAMİK VE ADAPTİF YENİ BİR YÖNTEM

Umut TOSUN
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Dr. Cengiz Çelik
Tez Yöneticisi: Prof. Dr. Özgür Ulusoy
Ağustos, 2007

Bilgisayar Bilimi uygulamaları, genellikle verinin etkin bir biçimde depolanması ve getirilmesi ile ilgilenirler. Geleneksel veritabanlarının iyi tanımlanmış yapısı, gereken sorgu nesnelerine *İlişkisel Veritabanı* paradigmasını kullanarak etkin bir şekilde erişmeyi sağlar. Fakat günümüzde görüntü, video, ses klibi ve metin dökümanı gibi yapısal olmayan ve karmaşık veri ile uğraşmanın zorluklarıyla karşılaşılmaktadır. Multimedya Veri Edinme, Veri Madenciliği, Görüntü Tanıma, Makina Öğrenmesi, Bilgisayar Görüsü, Biyomedikal Veritabanları karmaşık verinin etkin bir biçimde yönetilmesini gerektiren alanlardır. Karmaşık ve yapısal olmayan veri çoğu zaman iyi tanımlanmış parçalara bölünememekte ve tam bir eşleme sorguları tanımlamak için uygulanamamaktadır. Bunun yerine, kullanılacak bir sorgu nesnesi yada prototip nesne sağlayarak benzer nesneleri veritabanının getirmesini sağlayan *benzerlik araştırması* kullanılmaktadır.

Benzerlik Araştırması için bir popüler yaklaşım da veritabanı nesneleri arasındaki ilişkiyi vektör uzayında ifade ederek bu ilişkiye yaklaşmaya çalışmaktır. Literatürde vektör uzaylarındaki benzerlik sorgusunu destekleyen iyi bilinen indeksleme yöntemleri bulunmaktadır. Fakat bu yöntemlerin yüksek boyutlu veri için etkili olmadığı gösterilmiştir. Diğer bir yaklaşım ise indeksleme için *Metrik Uzaylar* modelini kullanmaktır. Metrik Uzaylar üçgensel eşitsizlik özelliği taşıyan bir uzaklık fonksyonu ile tanımlanırlar. Verinin iç yapısı ile ilgili varsayımlar olmadığı için yüksek seviyeli bir soyutlama sağlarlar ve daha fazla uygulanabilirliğe sahiptirler. Yüksek boyutlarda daha iyi performans sağladıkları da gösterilmiştir.

Daha önceki bir çok çalışma indeks yapısı oluşturulduktan sonra yeni nesne

eklenmesine izin vermeyen statik metotlara konsantre olmuştur. M-Ağaç, Slim Ağaç, DF-Ağaç, Omni bazı popüler dinamik yapılardır. Bu metotlar taşan düğümleri ayırarak ve ağaca B-Ağaç çeşitleri gibi yeni seviyeler ekleyerek artarak büyüyebilirler. Maalesef bu yöntemler AESA, LAESA, Spaghettis ve Kvp gibi sabit global pivot seti taşıyan düz yapılara göre çok daha kötü performans göstermektedirler. Sorgu nesnesi ve pivotlar arasındaki uzaklıklar hesaplanarak, veritabanının bir kısmı önemli olmaktan çıkarılır. Pivot sayısı daha fazla seçicilik sağlamak için kolaylıkla arttırılabilir ve daha iyi performans elde edilir. Fakat belli bir sorgu yarıçapı için optimum sayıda pivot bulunmaktadır ve çok fazla pivot kullanımı sorgu ve indeks oluşturma maliyetlerini arttırır. Yakın zamanda yeni veritabanı nesneleri eklenebilen ve dinamik bir şekilde yeni nesnelerin bazılarını pivot olarak seçerek ilerleyen LAESA varyasyonu Sparse Spatial Selection(SSS) takdim edilmiştir.

Bu tezde SSS yönteminin kümelenmiş ve bir uca toplanmış dağılımlar için yol açtığı temel problemlere değinilecektir. Gerçek veri gruplarında bu tür özellikler sıklıkla gözlemlenmiştir. SSS'in simetrik ve dengeli dağılımlar ayrıca özel sorgu yarıçapları için optimize edildiği gösterilecektir. Bu tezin ilk ana katkısı kümelenmiş yada bir uca toplanmış veride uygulanabilecek yeni bir pivot seçim yöntemi sunmaktır. İkinci katkı ise değişik sorgu yarıçapları için doğru pivot seçim sayısını bulmak olacaktır. Ayrıca sunulacak yeni indeksleme yönteminin nesne ekleme maliyetine yük getirmezken ağaç tabanlı uygulamalara göre çok daha iyi performans sağladığı gösterilmektedir. Bunun yanı sıra bu yeni yapı veritabanındaki populasyon artışlarına da üstün şekilde adapte olabilmektedir.

Anahtar sözcükler: Metrik Uzay, Metrik Erişim Metotları, Kvp, Hkvp, EcKvp, M-Ağaç, Slim-Ağaç, DF-Ağaç, Pivot, Uzaklık Hesaplaması.

Acknowledgement

I would like to express my gratitude to my supervisor Dr. Cengiz Çelik for his trust, encouragement and support throughout this thesis.

I would like to thank committee members Prof. Dr. Özgür Ulusoy, Asst. Prof. Dr. Çiğdem Gündüz Demir, Asst. Prof. Dr. Pınar Duygulu Şahin, Dr. Tansel Özyer for reading and commenting on this thesis.

I would like to thank my family, especially my mother and my father for supporting and believing in me throughout my life.

I acknowledge CyberSoft Information Technologies for supporting my MSc studies.

I would like to thank Öznur Aslan, Ulaş Aslan, Umut Orçun Turgut and Cemil Sezer from CyberSoft for their help and great moral support.

To My Family

Contents

List of Figures

List of Symbols and Abbreviations

AESA	: Approximating and Eliminating Search Algorithm
α	: Constant value for DBPP and SSS
d	: Distance Function
DBPP	: Distribution Based Pivot Promotion
fc	: Failure to Cut Probability
GH-Tree	: Generalized Hyperplane Tree
GNAT-Tree	: Geometric Near-Neighbor Access Tree
HKvp	: High Performance Kvp
kNN	: k Nearest Neighbor Query
Kvp	: k Vantage Points
LAESA	: Linear Approximating and Eliminating Search Algorithm
M	: Metric Space
Mvp-Tree	: Multiway Vantage Point Tree
p	: Pivot Object
P	: Pivot Set
PP	: Pivots to Process
PCAIPD-Avg	: Pivot Cut Approximation Based on Inter Pivot Distances-Average
PCAIPD-Root	: Pivot Cut Approximation Based on Inter Pivot Distances-Root
PCAGD	: Pivot Cut Approximation Based on General Distribution
PCAQPD	: Pivot Cut Approximation Based on Query Pivot Distances
PCAPP	: Pivot Cut Approximation Based on Pivot Performance
q	: Query Object
r	: Query Radius
S	: Set of Database Objects
SSS	: Sparse Spatial Selection
Vp-Tree	: Vantage Point Tree
X	: Domain of Objects

Chapter 1

Introduction

Database applications tend to involve complex, unstructured objects. Examples are multimedia data like images, videos [38], biochemical and medical data [38], text documents, fingerprints and DNA sequences. Similarity search is very crucial in these applications since such data can neither be ordered in a canonical manner nor meaningfully searched by precise database queries that would return exact matches. The objective in similarity search is to find a subset of objects from a data set S similar to a query object q. Traditional database methods exploit well defined structures. Various attributes of the objects are represented as independent dimensions. Contemporary databases include more complex and less structured data.

Current vector based solutions suffer from dimensionality curse [38]. They generally use too much space or work slower than naive algorithms. Metric space approach is reported to deal better with high dimensions than vector based methods. It is also a higher level of abstraction.

In this chapter, we will start by defining metric spaces, then we will define the type of queries that can be executed in this domain, and finally overview the index structures defined for metric spaces.

1.1 The Metric Space

A metric space M is defined as

$$M = (X, d) \tag{1.1}$$

for a domain of objects X and a distance function d. This metric space satisfies the following properties:

non-negativity

$$\forall a, b \in X, d(a, b) \geq 0 \tag{1.2}$$

symmetry:

$$\forall a, b \in X, d(a, b) = d(b, a) \tag{1.3}$$

identity:

$$\forall a, b \in X, a = b \iff d(a, b) = 0 \tag{1.4}$$

triangle inequality:

$$\forall a, b, c \in X, d(a, c) \leq d(a, b) + d(b, c) \tag{1.5}$$

Distance functions represent the closeness of objects in that domain to the query object. Distance measures can be discrete or continuous. An example of a continuous distance function is the Euclidian distance between vectors. The edit distance on strings is an example of discrete distance functions.

Some of the popular distance functions are Minkowski Distances [38], Quadratic Form Distance [19], Edit Distance [27], Tree Edit Distance [30], Jaccard's Coefficient [38], Hausdorf Distance [21] and Time Complexity Measure [26].

1.2 Overview of Similarity Queries

Similarity search is the process of classifying data objects with respect to their distances defined by d to a query object q. It is a kind of sorting or ranking of objects. Distance measure is used to define which data objects should be considered similar to the query object. In this section, we define basic types of similarity queries.

A *range query* $R(q,r)$ is defined as

$$R(q, r) = \{s \in S, d(s, q) \leq r\} \tag{1.6}$$

where $q \in X$ is the query object provided by the user, and r is the radius or the threshold value of the query. All objects around q within the distance r are retrieved by the query.

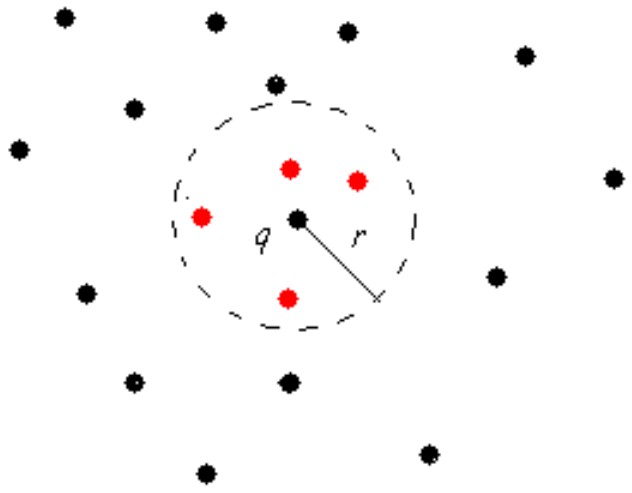

Figure 1.1: **Range query $R(q, r)$ with radius r and query object q.**

The result set of $R(q,r)$ can be ranked with their respective distances to query object q, in case of a need. Query object q need not exist in the collection $S \subseteq X$ to be searched. q belongs to the metric domain X. A real life example: Give me the group of towns that are within 50 km of Antalya [38]. Figure 1.1 shows a

range query.

Another type of similarity search in metric spaces is *nearest neighbor queries*. In its basic version, this query finds the closest object to the given query object q. A k nearest neighbor query, or kNN for short, finds the k nearest objects to q. If the number of objects to be searched k is larger than size of the database N, then we end up with the database as the result set. kNN is defined formally as follows:

$$kNN(q) = \{R \subseteq S, |R| = k \Lambda \forall a \in R, b \in S\text{-}R : d(q, a) \leq d(q, b)\} \qquad (1.7)$$

An example of k nearest neighbor query is: Select the three nearest cities to Antalya [38]. Figure 1.2 shows a kNN query.

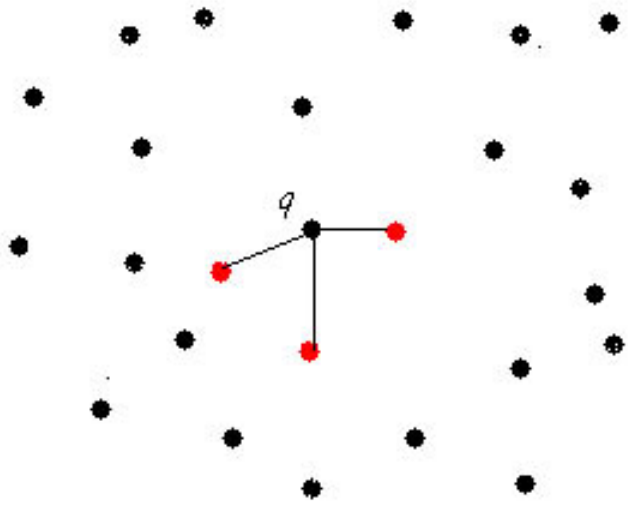

Figure 1.2: **Nearest Neighbor Query with $k = 3$**.

1.3 Overview of Metric Access Methods

The simplest way of performing a similarity search is to compare all objects of the database with the query object. However, computation of the distance function

is expected to be very expensive since we deal with complex objects. Therefore, research has focused on reducing the number of distance computations.

Index structures are used to reduce the number of distance computations. The index is built using the objects in the database. When performing queries, some of the objects are eliminated using triangle inequality without computing the distance to the query object.

Static indices are built using the whole collection, whereas dynamic methods allow insertion and deletion operations. Although some methods are implemented in secondary memory, studies show that the actual query times are either dominated by or in direct proportion to the number of distance computations. Therefore, we will use this criteria to evaluate the performance of our algorithms.

There are two major types of similarity search methods. These are *clustering-based* or *pivot-based* techniques. In pivot based methods, a subset of the objects are used as pivots. Index consists of the distances between each pivot and each object.

Given a range query, the distances from the query object to each pivot are computed, and then some objects are discarded without computing the distance, using the triangle inequality and the previously calculated distances. This operation is called *pruning*. Given $s \in X$ where s is an object of database X, p_i is a pivot and q is the query object to consider, the pruning criterion is described formally as:

$$|d(p_i, s) - d(p_i, q)| > r \tag{1.8}$$

Some examples of pivot-based methods are: Burkhard-Keller-Tree [7], Fixed-Queries Tree [3], Fixed-Queries Array [12], Vantage Point Tree [37] and its variants, Approximating and Eliminating Search Algorithm [36] and LAESA [28].

Clustering-based techniques divide the metric space to clusters, each having

a cluster center. A query may prune a region using triangle inequality and regional center. Examples of clustering based techniques are: Bisector Trees [23], Generalized-Hyperplane Tree [34], Geometric Near neighbor Access Tree [5] and Spatial Approximation Tree [29]. Algorithms of these methods may be found in the excellent surveys [13, 20].

1.4 Our Contributions

Global pivot based methods perform very well in terms of number of distance computations. They overperform tree-based structures because the number of total pivots are not limited by unrelated parameters like branching factor. However, in cases where there are a lot of pivots, these structures may spend too much time in computing distances to all pivots. The user of such structures have the option of calibrating a parameter that we call drop rate. The problem is, each query radius has a different optimal value of this parameter.

Another strong point of popular tree structures is that they are inherently dynamic. The global pivot based methods are static in nature, except for the recent structure, (Sparse Spatial Selection)SSS [6]. SSS solves two problems at once: how many pivots to keep for a particular database, and which new objects to promote as pivots. We will show that SSS is not designed very robustly in both of its missions under different distribution types and for different radius values.

Our first contribution is to devise a new method of automatically adjusting the drop rate. This way, even if the pivot promotion criteria erroneously promotes too much objects as pivots, or if the number of pivots is optimal for high radius values but too much for lower radius values; the structure can still avoid computing distances to some of the pivots.

The aim of second contribution is to avoid assigning too few pivots. For example, SSS fails in this fashion when the distribution is skewed toward high distance values. We will use a distribution sensitive method of deciding when to create a new pivot.

Currently all of the Kvp variants work in memory. However, most of the time, data repositories are huge in amount such that it is not possible to store all the structure in main memory. Traditionally disk-based structures allow insertion and deletion operations since recreating the entire index structure would be too costly. We believe that our work constitutes an important step in creating a disk-based version of Kvp.

The organization of the rest of the thesis is as follows: In Chapter 2, we will present a brief survey of the pivot based methods, followed by a discussion of general issues in new approaches to disk based centralized similarity searching in Chapter 3. In Chapter 4, we continue with the challenges to make HKvp dynamic and overview the pivot selection algorithms. In Chapter 5, we propose drop rate optimization techniques, our alternative pivot selection technique DBPP, and Dynamic HKvp. Finally, in Chapter 6 we present our conclusions and future work.

Chapter 2

Global Pivot Based Methods

In this chapter we concentrate on the global pivot based methods which improve query performance and construction cost when compared to AESA [36], LAESA [28], Spaghettis [11] structures.

Query processing in pivot based methods uses the pre-computed distances between database objects and pivots. A database object is eliminated without computing its distance to the query object if it can be classified as inside or outside the query radius by looking at its distances to the pivots. Pivots have different effectiveness in eliminating objects based on their distances to the query object. Prioritized vantage points(*vps*) [9, 10] is a new approach to improve the extra CPU overhead of pivot based methods. It only processes a promising subset of the pivots based on their distances to the query object. FQA[12] uses fewer bits to encode distance information between pivots and database objects. This causes reduction in pivot accuracy. Kvp [10] is an enhancement of prioritized vantage points. It organizes pivot distance data to reduce space requirements. It only stores the promising pivot distances which also means that there are less distances to process during a query, thus less CPU overhead.

Very few of the index structures use the advantage of more preprocessing costs to improve query performance. LAESA [28], Spaghettis [11], FQA [12] and Kvp [9, 10] are more effective than tree structures because of this fact.

HKvp is a structure based on Kvp which improves on LAESA. EcKvp [10] is a new structure which is based on the HKvp structure. It offers considerably lower preprocessing times with a small performance degradation. It uses a pivot index to decrease construction costs. Distances between the objects and pivots are retrieved by querying this pivot index.

2.1 Prioritized Vantage Points

Prioritized vantage points method [9, 10] stores $k \times n$ distance values, where n is the database size and k is the number of pivots. At a cost of few number of more distance computations, *vps* improves the CPU overhead of query processing.

Pivots are more effective when they are close to or far from the query object. Basic vantage points methods compute the distance between a query object and all the pivots and they process pivots in arbitrary order. Prioritized vantage points structure processes only close or far pivots. This approach does not add any extra burden to the process while decreasing the CPU overhead.

2.2 Kvp

It is desirable to use pivots that are particularly close to the query object. A pivot is more effective for objects that are close to or distant from it. Kvp [9, 10] finds such pivots, and keeps only the distances to these promising pivots. In priority vantage points approach, we do not know where the query object will be. Thus all pivot distances are kept. Kvp evaluates the distance relations between the pivots and database elements at construction time. It stores only the most promising distances. This reduces the CPU overhead while decreasing the space requirements.

There are two ways Kvp can be implemented. The first approach is the classical one where every pivot is resembled with an array of distances to database

objects. The object distances may be sorted to use binary search. Other approach, which is also used in implementation of this thesis is to have a collection of object entries, where each object entry stores the distances to to its selected pivots. The second approach is preferable since database insertions and deletions are much more easier to implement.

Kvp is very similar to classical pivot based methods except the way pivot distances are stored. It only stores a subset of the pivot distances. Query processing is the same with classical vantage point methods. Every object maintains a lower and upper bound for the distance to the query object. Pivots are used to tighten these bounds. The distance of an object to the query object is calculated if bounds are good enough to discard the object. An object is discarded if the bounds prove that it is within the query range or out of the query range. If the bounds do not satisfy the elimination of the object, the distance between the database object and the query object is computed.

As the number of pivots increase, query performance is improved by spending more time at the construction without increasing the space and CPU overhead. In spite of the fact that it uses less space than priority vantage points, Kvp ends up with a very similar query results. CPU overhead and space reduction is closely related in Kvp.

Even though Kvp structure works in main memory, it is easily adaptable to the disk. Kvp requires sequential scan of distance values rather than a binary search unlike some of the other pivot based methods like Spaghettis [11].

When the optimal number of distance computations is lower than the number of pivots used by Kvp, we face with a problem since Kvp computes distances to all pivots. HKvp [10], overcomes this problem.

To sum up, pivots which are close to or distant from the query object are more effective. Priority Vantage Points structure processes more promising pivots to reduce CPU overhead. Kvp improves this idea further. It only stores and processes some of the distances among database objects and pivots. There is a little performance penalty in terms of number of distance computations at query

processing but this is compensated by pivot prioritization scheme of Kvp which processes more promising pivots earlier.

2.3 HKvp

In this section, we introduce the structure HKvp which stands for "*High Performance Kvp*". The underlying working principle of global pivot based methods is that we have a very large database with a very expensive distance function. The number of pivots is assumed to be very limited with respect to the database size. There are exceptions to this assumption. Database size may be limited or application may require high number of pivots. The ratio of the number pivots to the database size may not be always low.

A typical pivot based structure begins search process by computing all distances between pivots and the query object. With the assumption that the probability of a pivot not eliminating an object is fc, after processing k pivots, there would still be fc^k objects that remain not eliminated. Hence, the total cost of a query is expressed by the equation:

$$Cost(q, r) = k + nfc^k \tag{2.1}$$

HKvp tries to find an optimum k value for a given query object and radius. Classical pivot based methods including Kvp fail to find the query result with a meaningful number of distance computations when the solution requires fewer number of distance computations than k. After an optimal number of pivots is reached, second part of the Equation 2.1 is dominated by the first part which is the cost of calculating pivot distances with the query object.

The value of the query radius effects the optimal number of pivots. Easier queries involving low dimensions or low query radii require fewer pivots than more difficult queries. A pivot based method may perform worse even though more effort is spent in construction because it has more pivots to process than

generally needed. HKvp is a structure performing significantly better in these kinds of queries. The major drawback of current HKvp is that it works with a drop rate parameter from user. In this thesis, we propose a group of methods to calculate the drop rate on query time.

Like AESA [36] and LAESA [28], HKvp also eliminates pivots as well as ordinary database objects. It reduces space complexity and processing times like Kvp.

HKvp uses both upper and lower bounds for object elimination unlike LAESA which uses only lower bounds for pivot and object elimination. HKvp chooses the next pivot to process while maintaining the distance bounds for pivots.

HKvp waits until all pivots are inside or outside the query range. When we have the best information about the bounds of a pivot, it chooses which pivots to have their distances to the query object calculated. HKvp does not discard the approximate pivot bounds of remaining pivots and they are also used in object elimination. HKvp has two phases as shown in Figure 2.1 where q is the query object, r is the query radius, P is the set of pivots, PP is the set of pivots to process and *resultSet* is the set of objects qualified for the query. The first phase computes distance bounds of the pivots. Some of these distances to the query object are computed exactly. Remaining bounds are just approximations based on distance relations with other pivots. In the second phase, more exact bounds are calculated with respect to drop rate and objects are visited using final bounds.

[1] Set all pivot bounds as [-∞, +∞] in *pivotBounds*
[2] $PP \leftarrow P$
[3] *finalBounds*$\leftarrow${}
[4] While PP is not {}
[5] p=most promising pivot of *pivotBounds*
[6] $d_{pq}=d(p,\ q)$
[7] If $d_{pq} \leq r$
[8] *resultSet*$\leftarrow$*resultSet*$\cup p$
[9] End if
[10] $PP=PP-\{p\}$
[11] Remove bounds of p from *pivotBounds*, put into *finalBounds*
[12] Forall $[L_j, U_j]$ based on d_{pq} and $d(j,\ p)$ where L and U are bounds
[13] Update $[L_j, U_j]$ based on d_{pq} and $d(j,\ p)$
[14] If $U_j \leq r$
[15] *resultSet*$\leftarrow$*resultSet*$\cup j$
[16] $PP \leftarrow PP-j$
[17] Else if $L_j > r$
[18] $PP \leftarrow PP-j$
[19] End if
[20] End for
[21] End while
[22] *ncompute*$\leftarrow$(1-dropRate)*times* $|$*pivotBounds*$|$
[23] For *ncompute* times do
[24] p=most promising pivot of *pivotBounds*
[25] $d_{pq}=d(p,\ q)$
[26] If $d_{pq} \leq r$
[27] *resultSet*$\leftarrow$*resultSet*$\cup p$
[28] End if
[29] $PP=PP-\{p\}$
[30] Remove bounds of p from *pivotBounds*, put into *finalBounds*
[31] Update bounds of other pivots
[32] End For
[32] Put the rest of the elements in *pivotBounds* to *finalBounds*
[32] Process the database objects like Kvp by using *finalBounds*

Figure 2.1: **HKvp Range Query**

2.3.1 Pivot Selection

HKvp computes distances to promising pivots. However, determining how valuable a pivot is a difficult task. It has been shown that a good performance is achieved by selecting the next pivot to process as the one with the lowest lower bound for the distance to the query object [28]. After the chosen pivot has its distance evaluated with the query object, bounds of other pivots are improved. [10] explores two new approaches in addition to this approach. It selects the pivot with the highest bound, and the pivot having the greatest distance between lower and upper bounds. According to [10] wide pivots give the best results and far pivots give the worst results.

2.3.2 Drop Rate

The first phase of the HKvp range search algorithm processes pivots until every pivot is either processed by getting its distance to the query object computed, or is proved to be outside the query range. If all the objects were also pivots like in AESA, that would be the optimal ending point for the query. However, there are some ordinary database objects that could be pruned if we had some extra distance information about the pivots. Because of this, HKvp chooses a group of pivots from the eliminated pivots for further processing. The decision for the second phase is controlled by a parameter called *drop rate*. After the first phase is executed, a set of P^* pivots have their exact distances to query object calculated. The rest of the pivots P^{**} only know their bounds approximately. To obtain better bounds for pivots, we continue to process the remaining set P^{**} by restricting the number of pivots to process with a drop rate value. For a fixed number of database objects, as the number of pivots increase, more of them should be dropped for better performance. This is because the probability of a pivot to be useful for object elimination decreases when there are less objects per pivot.

2.3.3 Pivot Limit

The pivot limit is the parameter of Kvp which determines the number of pivot distances stored and used per database object. HKvp also uses this parameter. Using pivot limit to store distances reduces the space complexity and CPU overhead at the same time.

2.4 EcKvp

EcKvp [10] has the advantages of low space and query time like Kvp while providing lower preprocessing time with a small degradation in performance. It stores the pivots in an index structure. Rather than computing the distance values between and object and all the pivots, it retrieves the relevant distances by querying the inner index. This way, it reduces the construction cost.

EcKvp permits the use of a greater set of pivots. This makes the drop rate parameter even more important.

Chapter 3

Dynamic Methods

Many of the current structures are memory-based. However, to process larger volumes of data, structures using disk are needed. Handling data files that can change size dynamically is a difficult task. Almost all of the dynamic methods in literature are tree-based. These structures split nodes to make room for new insertions. Splitting a node forces the algorithms to update the information used in pruning. To keep the splitting costs down, these structures keep limited information in the nodes. For example, GNAT is a static structure that is similar to the dynamic ones we will introduce in this chapter, except it keeps more detailed information in the nodes that enables it to perform better at the query time.

3.1 M-Tree

M-Tree [14] is a dynamic and disk-based structure. It is feasible and purposeful to use M-Tree in frequently modified databases due to its deletion and insertion capability. With a bottom-up approach represented as a balanced tree, it handles insertion and deletion operations efficiently. M-Tree have several variations such as Pivoting M-Tree [31], M^+-Tree [39] and M^2-Tree [16] are the most reputable ramifications of M-Tree idea. M-Tree is also a starting point for Slim-Tree and

DF-Tree structures.

M-tree is similar to the R-Tree in the way it organizes its nodes as disk pages, and the splitting algorithm to keep the tree balanced. Rather than defining sub-tree regions as hyper-rectangles, it uses a central representative object and a radius around this object.

Each internal node of the M-Tree carries representative objects, the radius values that define the sub-tree regions, and pointers to the sub-trees. Leaves are the nodes where the objects are stored. Minimum bounding rectangles cover the borders of the subtree in R-Trees. This information is stored for each subtree in leaf nodes. M-Tree is a metric space structure. Metric space structures do not have coordinate systems. Thus, we can not define this kind of information in a non-leaf node. M-Tree uses a covering radius to form a ball region bound like R-Tree.

Pivots are key elements of M-Tree. In M-Tree all objects are stored in leaves. The same object may be at leaves or internal nodes as a pivot at various time intervals because of the dynamic structure of the M-Tree. The fanout of M-tree is based on the page size and the size of the objects.

Figure 3.1: **Atomic Structure of an Internal node of the M-tree.**

Figure 3.1 shows the atomic structure $\langle p, cr, d(p, pp), p^* \rangle$ of an internal node, p is a pivot and cr is the covering radius around p. The parent pivot p is denoted as pp and $d(p, pp)$ is the distance function between p and the parent pivot pp. p^* is the pointer to the child subtree. All objects of the subtree pointed by p^* are at a maximum distance of cr from pivot p. $d(s, p) \leq cr$ as a general fact. Storing

the distances to parent nodes increases the elimination in search process.

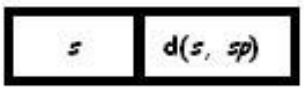

Figure 3.2: **Atomic Structure of a Leaf node of the M-tree.**

Figure 3.2 shows the atomic structure of a leaf node. A leaf node consists of objects formally: $\langle s,\ d(s,\ sp)\rangle$. Here, s is a database object and $d(s,\ sp)$ is the distance between s and its parent object.

3.1.1 Insertion Algorithm

Covering radius for a corresponding subtree is not always the minimum value it should be. Thus, bounding ball regions of the nodes intersect. Using the minimum values of respective bounding ball regions result in a more efficient search since ball regions of the nodes become disjoint. The original M-tree does not consider bulk loading. A bulk load algorithm has been proposed in [14]. However this technique is based on optimizations to tree construction. The performance boost is not trivial.

The insertion algorithm of the M-Tree looks for the most suitable leaf node to insert a new object. The tree is build adaptively as new data objects are inserted thanks to dynamic structure of M-Tree.

The insertion algorithm of the M-Tree behaves as follows:

• Traverse the tree down until a subtree for which the covering radius cr contains the inserted node, i.e., $d(s_N,\ p) \leq cr$.

• If more than one subtree exists with covering radius cr containing the inserted node, the one which has its pivot closest to the inserted object s_N is chosen. This supports the idea of minimizing the covering radius.

• If there is not a pivot which contains the inserted object in its covering radius, then choose the pivot which needs the minimum increase for cr to cover the area containing the previous objects and the newly inserted object. Traverse down the tree until a leaf node, in this way. Adjust the affected radii of nodes during the traversal.

• If insertion into a leaf causes an overflow: Allocate a new node N^* at the same level with node N sharing the *objectCount+1* entries. Select new pivots as p_N, p^*_N . There are efficient algorithms for this pivot selection in [14].

3.1.2 Range Search Algorithm

Insertion methodology of the M-Tree tries to minimize the intersecting regions of the ball regions. This is important for range search $R(q,\ r)$. The Range Search Algorithm for M-Tree is as follows:

• Let the current node N be an internal node, consider all non-empty entries $\langle\ p,\ cr,\ d(p,\ pp),\ ptr\rangle$ of N.

• The lower bound for the distance $d(q,s)$ is $\mid d(q, pp) - d(p, pp) - cr \mid > r$. If the lower bound is greater than the query radius r, the entry is eliminated without any distance computation. Therefore the subtree need not be considered.

• If $\mid d(q, pp) - d(p, pp) - cr \mid \leq r$ holds, distance $d(q,\ p)$ should be calculated. Having the value of $d(q,\ p)$, some of the branches are eliminated by: $d(q, p) - cr > r$.

• Recursively search the non-eliminated entries.

• Each leaf node entry $\langle\ s,\ d(s,\ sp)\ \rangle$ is eliminated if $\mid d(q, sp) - d(s, sp) \mid > r$. If the entry cannot be eliminated, the distance $d(q, s) \leq r$ is calculated.

3.1.3 Algorithm Complexity

Let n be the number of distances occupied in leaf node, m_N be the number of internal nodes, and each node has a capacity of m entries.

The the space complexity of the M-Tree is O($n+mm_N$) distances.

The construction complexity is O($nm^2 \log_m n$) in terms of number of distance computations.

3.2 Slim-Tree

Slim-Tree [32] aims to reduce the intersection of ball regions. Slim-Tree is an extension of M-Tree. It improves M-Tree idea for insertion and node splitting while improving storage efficiency. It changes the splitting methodology of M-Tree with a more compact way. The structure of the Slim tree is the same as that of the M-tree.

3.2.1 Insertion Algorithm

Slim-Tree Insertion Algorithm tries to locate a suitable node to cover the newly inserted object starting from root. The node whose pivot is nearest to new object, is selected in case of not finding a node. In spite of the fact that M-Tree chooses the node whose covering radius cr requires the smallest enlargement, Slim-Tree chooses the node whose pivot is nearest to the new object. In case of a tie break, Slim-Tree selects the node which exploits the minimum space. In M-Tree, this procedure is as selecting the one whose pivot is closest to the new object.

The modified insertion strategy of Slim-Tree aims at filling all the empty nodes first. In this strategy, the splitting procedure is postponed until it is inevitable. It boosts the node utilization, cuts the number of tree nodes needed to organize the database. With this strategy, I/O costs for Slim trees are dramatically decreased

while the number of distance computations are nearly the same for both M-Tree and Slim-Tree. The same fact applies to query execution. However most of the time I/O cost is not an issue when compared with the cost of distance computation. Slim-Tree is also motivated on reducing the relatively high construction costs of M-trees. The split algorithm of the Slim-Tree is successfully used for clustering. The construction is based on the minimum spanning tree algorithm.

Slim-Tree splitting algorithm is summarized as follows:

[1] Minimum Spaning Tree Construction
[2] Removal of Longest Edge
[3] Content of the New Nodes arise as the resulting subgraphs
[4] Selection of a Pivot for Each Group as the Object with the Shortest Distance to All Other Objects Respectively

Figure 3.3: **Slim-Tree Splitting Algorithm.**

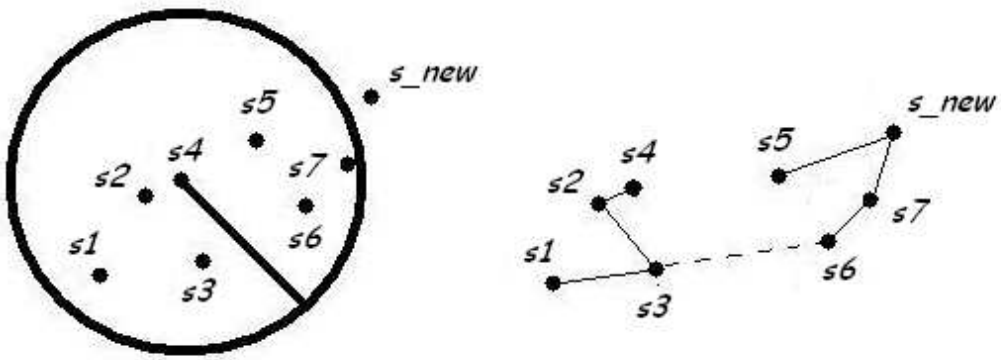

Figure 3.4: **Slim-Tree Splitting Algorithm.**

Figure 3.4 shows how the slim-tree splitting algorithm works. A newly arrived object *s_new* causes the node with pivot *s4* to split. The longest edge to be removed is shown as a dashed line on left of the figure. Using the minimum spanning tree algorithm two clusters occur with pivots *s2* and *s7*

Most of the time everything is not as easy like the example in figure 3.4. The problem with the slim tree splitting algorithm is that, it does not guarantee a balanced split. The work presented in [32] suggests choosing a group of appropriate edges and selecting the edge to be removed by looking at the balance of the clusters. However, this is not a solution in all cases. Moreover, slim-down algorithm may end up with a deadlock [32].

Experiments in [32] compare the efficiency of the new Slim-Tree splitting strategy with the original M-Tree splitting strategy. The results claim that the query execution times remain the same while construction is much more faster than M-Tree in Slim-Tree. This results in a fact that dynamically changing environments should use Slim-Tree because of the high splitting costs.

3.2.2 Algorithm Complexity of Splitting Algorithms of M-Tree and Slim-Tree

It is reported in [14] that M-Tree has a splitting complexity of $O(n^3)$ and $O(n^2)$ distance computations. The splitting algorithm of Slim-Tree is based on minimum spanning tree. This algorithm needs $O(n^2)$ distance computations and the total execution time is $O(n^2 \log n)$. Slim-Tree splitting algorithm suggests a fully connected graph with n vertices and $n(n\text{-}1)$ edges, where each edge is given a weight of distance between a pair of connected objects.

3.3 Omni

Designing a database system such as M-Tree [14] or Slim-Tree from scratch is a complex task. Omni [18] proposes a family of alternative methods to improve existing techniques. It is easy to implement on top of systems like M-Tree [14], Slim-Tree [32], R-Tree [22] and sequential scan. It suggests an indexing structure by selecting a set of objects as foci and fix the other objects distances to this set. The foci set is a dynamic structure and it is updated regularly with database alterations.

The foci increase the pruning during the query processing. An index structure storing the array of distances from each object to the foci reduces the triangular inequality comparisons either. Omni [18] shows a good performance with growing database sizes.

By an inexpensive algorithm, Omni chooses an adequate number of objects to be used as foci. It aims to give the optimum memory requirements while decreasing the number of distance computations.

3.3.1 Omni Concept

Omni [18] uses a set of global foci to prune distance calculations. It may be used either alone or with an existing metric access method. The elements of the Omni concept are presented as follows:

Definition 1: Given a metric space $M=\langle X, d\rangle$. Let N be the number of objects in the database, k be the number of neighbors in kNN query, l be the number of foci. **Omni Foci Base** is the set $F = \{f_1, f_2, ..., f_l \mid f_k \epsilon X, f_k \neq f_j, l \leq N\}$ where each f_k is a focus of X.

Definition 2: Given the **Omni Foci Base** F and an object $s_i \epsilon S$, **The Omni Coordinates** C_i of an object s_i is defined as the set of distances from s_i to each focus in F. Thus, **The Omni Coordinates** of an object s_i is defined

formally as $C_i = \{\langle f_k, \mathrm{d}(f_k, s_i)\rangle, \forall f_k \epsilon F\}$.

Each newly inserted object has its Omni Coordinates evaluated and stored. Omni Coordinates are used to prune distance calculations through the triangular inequality property. Use of Foci causes two kinds of costs: costs of data structure and calculation of Omni Coordinates for each object in S. Assuming the usage of Omni over a disk based technique storing the set of objects and foci, the memory cost of the structure is compensated. However, disk I/O cost is an issue. Increasing disk access slows down the query processing. Omni tries to give the best tradeoff with an optimal usage of foci and disk storage. Moreover, decreasing the number of extra distance calculations in query processing pays off. Complex and large objects, such as images and audio, need huge memory. The space needed to keep a few extra distance calculations is relatively insignificant. Exemptions may occur in Omni Coordinates for smaller objects.

All in all, Omni prunes distance calculations while compensating the increasing disk accesses. Moreover, implementation costs of Omni are very low.

3.3.2 Omni Foci Base

As discussed in the previous subsection, we know that there is a tradeoff between the number of foci used and the space and time spent to process them. Concept of Minimum Bounding Region is proposed to meet the maximum gain while using a minimum set of foci.

Definition 3: Given the **Omni Foci Base** $F = \{f_1, f_2, \ldots, f_l\}$ and a collection of objects $S = \{s_1, s_2, \ldots, s_n\} \subset X$; **Minimum Bounding Region** of S is defined as the overlapping metric intervals $R_A = |_1^l I_i$, where $I_i = [\min(\ d(s_j, f_i)\ ), \max(\ d(s_j, f_i)\ )]$, $1 \leq i \leq l$, $1 \leq j \leq n$.

Each focus defines a metric sub-space. This is called a ring. A Minimum Bounding Region is the subset of S such that the Omni Coordinates identify as including the answer of a query. That is the region where foci can not prune the objects. The result set is always included in the Minimum Bounding Region.

However, there are false positives. Thus, a final refinement is crucial. In this step, distances are calculated respectively.

In spatial databases, it is claimed in [18] that the intrinsic dimension defines a limit for the appropriate number of foci. It is proposed in [18] that twice the number of intrinsic dimension of a database is suitable as foci count. More than a count twice the intrinsic dimension of the database leads a negligible reduction in the Minimum Bounding Region. Even a count of one or two may be sufficient for foci count. A minimum reduction for two foci is satisfied when they are orthogonal. However, it is not possible to fix the foci as orthogonal since they are previously defined. One more focus might be used when foci are not apart from each other to distribute the load of not ideally distributed foci. By this sense, a good number for the cardinality l of F is claimed to be between the next integer that contains the intrinsic dimension $\lceil D_{\mathcal{Q}} \rceil + 1$ and $2^{*}\lceil D_{\mathcal{Q}} \rceil + 1$. Not only this formula is used for spatial data sets, but also repeated for metric data sets. If we generalize this formula, data sets with $1 < \lceil D_{\mathcal{Q}} \rceil \leq 2$ will lead to three foci(equilateral triangle); data sets with $2 < \lceil D_{\mathcal{Q}} \rceil \leq 3$ will lead to four foci(tetrahedron) etc.

3.3.3 Omni HF Algorithm

In this subsection, we describe the implementation issues of foci selection. Let N be the number of database objects and l be the number of foci. The foci selection algorithm, which is called HF-Algorithm has the complexity $O(\frac{N!}{(N-l)!})$. The algorithm effectively uses $O(N)$ distance calculations. It tries to find a subset from the database that leaves the other objects inside the region surrounded by foci.

The algorithm starts with randomly choosing an object s_1. Later, it starts searching a pair of objects far enough. The first focus is the furthest object from s_1. Consequently another focus is chosen as second and distance btw. the first and second focus is stored. The algorithm continues choosing foci with most similar distances to previously choosen ones. This operation uses an error

function $error_i = \sum \mid edge - d(f_k, s_i) \mid, \forall k \epsilon F$ to select foci. Minimization of error function is crucial to select foci. HF-Algorithm in figure 3.5 finds the foci set F of cardinality l of a data set S.

[1] Choose an $s_i \epsilon S$ randomly.
[2] The object f_1 furthest from $s_i \epsilon S$ is selected.
[3] Add f_1 in F.
[4] The object f_2 furthest from f_1 is selected.
[5] Add f_2 in F.
[6] $d(f_1, f_2)$ is stored as an edge.
[7] Use the edge to calculate $error_i$.
[8] While count of foci < count of foci to be found:
[9] For each $s_i \in S$, $s_i \notin F$:
[10] Calculate $error_i$
[11] Select $s_i \in S$ such that $s_i \notin F$ and $error_i$ is minimal.
[12] Insert s_i in F.
[13] End For.
[14] End While.

Figure 3.5: **HF-Algorithm**

3.3.4 Omni Sequential

Sequential scan over omni concept is called Omni Sequential. A range query $R(q, r)$ with radius r is operated by calculating each distance $d(f_k, s_q)$ from the query object s_q to each foci by creating omni coordinates. If $\mid d(f_k, s_i) - d(f_k, s_q) \mid > r_q$ for any of the $f_k \in F$, then the distance computation is eliminated. The k-nearest neighbors algorithm is similar to range query.

3.3.5 Omni B-Tree

B-Tree over omni concept is called Omni B-Tree. Omni B-Tree stores omni coordinates in l B-Trees. Each B-Tree has one focus and each query retrieves $I_k \subset X$ which is used to define the minimum bounding region. Each I_k in range

$r_{min} = d(f_k, s_q) - r_q$ and $d(f_k, s_q) + r_q$ is retrieved. The answer set is the number of non-eliminated objects after the distance calculations of s_q and objects in the intersections. Showing a similar behaviour to range query, *kNN* is used with radius estimation techniques.

3.4 DF-Tree

Index structures make complex data retrieval easier. They organize data to eliminate needless comparisons during queries. DF-Tree [33] defines a new measurement of prunability and proposes a new access method to minimize the number of distance computations required to answer a query. DF-Tree uses most of the properties of Slim-Tree. It uses the foci concept of Omni over Slim-Tree and defines an adaptive structure by When To Update/How To Update algorithms to dynamically modify the global representative set with distorted elimination power because of database alterations.

3.4.1 DF-Tree Basics

In tree structures, data is stored in nodes with fixed capacity using a reference object for each node to represent other objects. Previous distance computations are stored either in the tree and there are representatives in the tree. Triangular inequality is used as other metric structures to prune distance computations. Figure 3.6 shows the DF-Tree structure constructed from a database shown in Figure 3.7. The root node does not have a representative and the data set S is stored in the leafs.

Covering radius is also an important issue in DF-Tree as in M-Tree [14] and Slim-Tree [32]. Covering radius *cr* for leaf nodes is defined by choosing one of the objects $s_j \in S$ stored in a leaf node i as representative and calculating every distance between representative and every object stored in the node. The largest distance is set as *cr*. This means that no object with a further distance to the

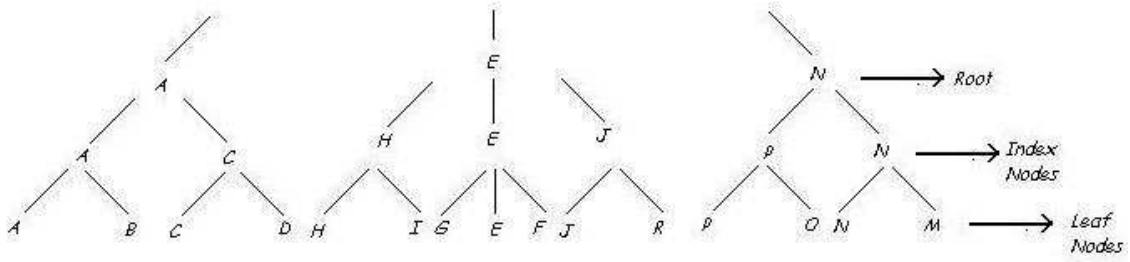

Figure 3.6: **DF-Tree Visualisation of The Sample Database.**

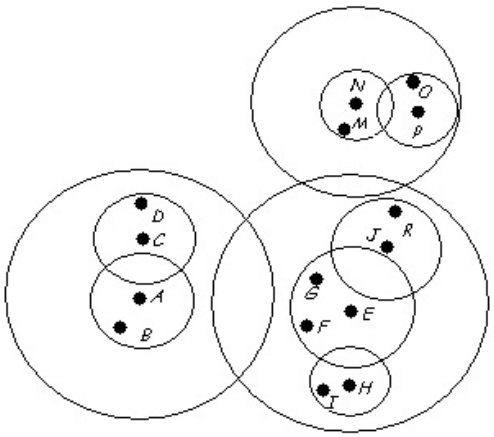

Figure 3.7: **Sample Database.**

representative than cr may be found in node i. Covering radius of the remaining nodes other than leaf nodes are calculated similarly: the distance between its representative and the furthest object of the node plus the covering radius of the node where this object is representative.

Considering a range query of center s_q and radius r_q, every node i with representative s_{Ri} can be pruned if one of the following two criterias is satisfied. Thus, the triangular inequality enables pruning both on traversal of subtrees in non-leaf nodes and on distance computations among the query object and objects in the leaf nodes. The criteria are as follows:

$$d(s_{Rep}, sq) + r_q < d(s_{Rep}, s_{Ri}) - cr \tag{3.1}$$

$$d(s_{Rep}, s_q) - r_q < d(s_{Rep}, s_{Ri}) + cr \tag{3.2}$$

The same concept applies at leaf nodes with a difference. The value of cr is zero in leaf nodes. With this property, DF-Tree enables pruning on subtrees in non-leaf nodes as well as on leaf nodes.

DF-Tree is a dynamic structure, because each new object is inserted in a node that is able to cover it. If a newly arriving object is covered by any of the objects, the node which requires a minimum enlargement of covering radius is selected to insert the object. Like in M-Tree [14] and Slim-Tree [32], if the node capacity is exceeded the node splits and new representatives are chosen.

Using more than one representative improves pruning ability. Larger portions of the database is eliminated by using two or more references. However using multiple representatives is not so easy and may lead to static structures. The structure becomes static if the set of reference objects of a given node defines in which of the descending subtree the new object should be stored.

Whenever a reference is altered, the objects stored in a given subtree must be moved to another subtree. In order to be dynamic, each object should be stored in more than one place or store each object to select the representatives in a bottom up approach while satisfying the reference object changes without the effect of the upper levels of the tree but its subtree. Both alternatives have problems. Allowing for more than one place to store each object causes more effort to answer a query. On the other hand, choosing the representatives in a bottom up approach prevents the combined effect of a set of representatives along the path of nodes to that node.

The Slim-Tree [32] and the M-Tree [14] choose a compromise of the two structures. Slim-Tree [32] solves the first problem above minimizing intersecting node regions of M-Tree [14]. DF-Tree [33] also solves the second problem using more representatives dynamically.

3.4.2 DF-Tree Structure

The proper use of Global Representatives in a metric tree reduces the number of distance computations required to answer a query. Single representative of each node of the tree is called as node representative and global representatives are defined formally as follows:

Definition 4: Let $M = \langle X, d \rangle$ be a metric space, where X is the domain of objects and d is a distance function. In a dataset $S \subset X$ with N objects, a **Global Representative Set** is $G = \{g_1, g_2, ..., g_p \mid g_k \in S, g_k \neq g_j, p \leq N\}$ where each g_j is a Global Representative, and p is the number of global representatives contained in G.

Each representative is independent from others and applied to every database object. Each global representative defines a distance field DF over the domain X. DF-Tree builds a tree with the use of node representative per node as in Slim-Tree with the addition of distance fields of global representative set G. Distance fields do not have a role in tree construction. Global representatives may be selected at any time before the answer of first query by calculating the distances of global representatives to each object. DF-Tree structure consists of data used to build the tree and the distance field attributes. DF-Tree [33] components resemble Slim-Tree [32] and algorithms are very similar.

3.4.3 Prunability

The number of distance computations to be pruned depends on sizes of areas defined by each representative respectively, query center and representative radii. A new concept of prunability is defined as follows in DF-Tree:

Definition 5: Let Q be a set of similarity queries over a tree, $Nt_b\ (q_i)$ be the total number of not pruned objects in node b during query process q_i ,$Nu_b\ (q_i)$ be the number of objects in node b that actually qualify to answer the query q_i. **Prunability** $P_h(Q)$ is the average of the relation $Nu_b\ (q_i)$ / $Nt_b\ (q_i)$ applied to

each node b accessed at a given level h to answer each query $q_i \in Q$. To check whether an object is pruned by distance field, algorithm in Figure 3.8 is used.

[1] For each global representative $g_j \in G$
[2] Set g_j as representative s_{Rep},
if Equation 3.1 or Equation 3.2 holds object is pruned.
Thus return true, otherwise continue.
[3] End For
[4] return false

Figure 3.8: **Prunability Check for an Object**

3.4.4 Range Query Algorithm

Range query of DF-Tree starts looking at the root node of the tree. The representative is set to s_{Rep}. The distance between an object and the representative is computed. If Equation 3.1 or Equation 3.2 do not apply, global representatives are tried using algorithm in Figure 3.8. If none of the equations hold for none of the representatives(node or global) distance computation is inevitable. Otherwise the subtree can be pruned from database. The range search algorithm to process a subtree is shown in Figure 3.9:

[1] Calculate $d(s_q, s_{Rep})$
[2] For each subject s_j for node i
[3] Set s_{Rep}
[4] If Equation 3.1 or Equation 3.2 holds continue
Else call Algorithm of Figure 3.8
[5] If Algorithm in Figure 3.8 returns true then continue
[6] If s_j is a leaf node, put it in result set.
Else process the subtree.
[7] End For

Figure 3.9: **DF-Tree Range Search Algorithm**

3.4.5 Nearest Neighbour Algorithm

kNN Algorithm uses a priority queue Pr of size k to store the distance of each candidate object to the query object. The distance of the furthest object is set to current query radius r_c. The algorithm starts with Pr empty and r_c is set to infinity until there are k objects in the queue. A new object is inserted if its distance with query object is smaller than r_c. The first phase of the algorithm stores the unprocessed objects in a priority queue Pw. The objects in Pw are processed in the second phase. The algorithm lasts until Pw is empty. The k nearest neighbor search algorithm to process a subtree is shown in Figure 3.10:

- PHASE 1 -

[1]If node i is a non leaf node:
[2] For each object s_j of node i
[3] Set s_{Rep} as the representative
[4] If Equation 3.1 or Equation 3.2 holds, continue
[5] Else if it returns true after calling Algorithm 3.8, continue
[6] Else insert s_j in Pw
[7] End For
[8]End If

- PHASE 2 -

[9]While Pw is not empty, get s_j and r_j from Pw
[10] Set s_{Rep} as s_j
[11] If Equation 3.1 or Equation 3.2 holds, continue
[12] Else if the node is an internal node, process the subtree of object s_j
[13] Else
[14] For each object s_j of node i
[15] Set s_{Rep} as the representative
[16] If Equation 3.1 or Equation 3.2 holds, continue
[17] Else if it is true after calling Algorithm in Figure 3.8, continue
[18] Else insert s_j in Pr, arrange new r_c
[19] End For
[20]End While

Figure 3.10: **DF-Tree *kNN* Algorithm**

Chapter 4

Pivot Selection Techiques

Pivot based methods like Kvp and HKvp are not dynamic in nature and do not allow insertions and deletions. In this thesis, we aim to use Kvp and HKvp in a way that they can dynamically adapt themselves to efficiently process the new objects that might be marginally different than the existing objects. However, this is only possible with pivot promotion techniques that use some of the inserted objects as new pivots. There is some previous work in literature that selects a subset of the whole database as pivots resulting in a static structure. Ideas and observations from these static algorithms should be mostly valid for dynamic pivot promotion case. In this section, we discuss a general overview of the pivot selection schemes to be considered for pivot based methods.

Proximity search algorithms generally select pivots at random. However, the way the pivots are chosen drastically effects the algorithm performance. Consider two sets of pivots at the same size. The better chosen group can largely reduce number of distance computations while requiring much less space. The same situation may apply for two sets where one group is larger than the other. Thus, pivot selection is an important issue in pivot based methods.

The distances to be considered in this thesis are assumed to be expensive to compute such as comparing two fingerprints or color histograms. In many applications, distance computation is so expensive that it dominates I/O costs

and extra CPU time spent during similarity search process. For this reason, in this thesis the complexity of the algorithms will be measured in terms of the number of distance computations performed. Proximity search algorithms construct an index of the database and perform queries using this index. The algorithms are mostly based on the use of pivots [18]. Savings from distance computations are obtained by using these pivots with the concept of triangular inequality while answering queries.

Even though search algorithms are based on pivots to improve the performance, almost all proximity search algorithms based on pivots choose them randomly [8]. It is a well known fact that search performance is effected from selection of pivots. Heuristics to choose pivots better than random only try to choose objects that are far from each other. In spite of the fact that good pivots are outliers, selecting pivots as outliers do not guaranty the best pivot set [8, 38].

In this chapter, we present the most popular pivot selection techniques proposed in [8] and we will discuss the Spatial Selection of Sparse pivots SSS [6] which we will improve for our HKvp purposes. From the proposed techniques the best results are obtained by SSS [6].

4.1 Selection of N Random Groups

In N groups method[8], N random groups of k objects are selected from the database S. Their mean μ_d is calculated for each group of pivots. The group with the maximum mean μ_d is selected as the pivot set. The optimization process has a cost of $2kAN$ where A is the pairs of objects selected at random. The basic assumption to obtain the value of μ_d is as follows: A pairs of objects are randomly chosen as $\{(a_1,\ a_1^{'}),(a_2,\ a_2^{'})...(a_A,\ a_A^{'})\}$ from database S. All the pairs of objects have their distances with pivot set as $\{d_1,\ d_2,\ ...\ d_A\}$. The value of μ_d is estimated as $\mu_d = \frac{1}{A}\sum_{1\leq i\leq A} d_i$. This means $2k$ distance computations are incurred for each pair of objects to calculate d. Thus, μ_d is estimated by $2kA$ distance computations.

4.2 Incremental Selection

In incremental selection method[8], first a pivot p_1 is selected from a sample of N objects from database S such that the pivot alone has the maximum mean μ_d. Then a second pivot p_2 is chosen such that current pivot set $\{p_1, p_2\}$ has the maximum μ_d value. The process is repeated until k pivots are chosen. The optimization process has a cost of $2kAN$. Since the distances $d_{\{p_1,...,p_k\}}([a_r],[a_r^{'}]), 1 \leq r \leq A$ are kept in an array, the distance computations to estimate μ_d are not evaluated again when the i^{th} pivot is added. Only $d_{\{p_1,...,p_i\}}([a_r],[a_r^{'}]), 1 \leq r \leq A$ is calculated which is also expressed as $\max(d_{\{p_1,...,p_{i-1}\}}([a_r],[a_r^{'}]), d_{\{p_1,...,p_i\}}([a_r],[a_r^{'}]), 1 \leq r \leq A$. All in all, only $2NA$ distance computations are performed when a new pivot is added. Since there are k pivots, the total optimization cost is $2kAN$ distance computations.

4.3 Local Optimum Selection

The matrix $M(r,j) = d_{p_j}([a_r],[a_r^{'}])$ for $1 \leq r \leq A; 1 \leq j \leq k$, is constructed from k randomly chosen pivots and random objects of the database where A is the set of object pairs and μ_d can be estimated from $d([a_r],[a_{r'}]) = \max_{1 \leq j \leq k} M(r,j)$ for every r. The two largest values of each row of M, r_{max} and $r_{max'}$ are considered. The contribution of a pivot p_j is expressed by $M(r,r_{max})$ - $M(r, r_{max'})$. The contribution of the pivot p_j is defined as the sum of how much $d([a_r],[a_{r'}])$ increases in value due p_j for A rows. One of the pivots is selected as victim whose contribution is minimum to μ_d. It is replaced by a better pivot from a sample of X objects of the database. This process is repeated N' times.

The construction cost is $2Ak$ distance computations. Selecting a better pivot from X objects has a cost $2AX$ while search cost of victim is 0 since all the information is possessed by matrix M. When this operation is repeated N' times total cost is $2A(k + N'X)$. Considering $kN = k + N'X$, $N'X = k(N-1)$ the optimization cost is $2AkN$ distance computations. Using $(N' = k) \wedge (X = N-1)$ is called local optimum A where (N $'$=N-1 $\wedge$ (X =k) is called local optimum B.

4.4 GNAT's Pivot Selection

Although a tree-based structure, GNAT employs a pivot selection algorithm to decide which subset of the objects covered by a node as pivots. The algorithm works on a sample subset of the objects, but there is nothing that would prevent applying the same algorithm on the whole set except for efficiency considerations.

GNAT algorithm first selects a random object and works incrementally. At each step, it chooses the object that maximizes the minimum distance to the current set of pivots. This guarantees that the new pivot is as far away from the pivots as possible.

4.5 Spatial Selection

Sparse Spatial Selection (SSS)[6] is a pivot selection method that is based on the GNAT pivot selection algorithm. According to the experiments in [6], SSS is more efficient than previously defined methods. It adapts itself to the dimensionality of the database and the number of pivots to be used is defined by the method itself. It allows object insertions and deletions unlike the previously defined techniques. It is suitable for secondary storage with these properties.

The pivot set starts with first inserted object of the database. Let $(X,\ d)$ be a metric space, $S \subseteq X$ an object collection, and $M = \max\ d(s,\ s^*);\ s,\ s^* \in S$. Then an object is selected as a pivot if and only if its distance to any pivot in the current set of pivots is equal to or greater than $M\alpha$ where α is a constant around 0.4. This constant is obtained experimentally [6].

An object in the database is chosen as a new pivot if it is located at more than a fraction of maximum distance with respect to all current pivots. The SSS algorithm is formalized in Figure 4.1.

The value of α determines how far the pivots are from each other. Since the selected pivots are all more than $M\alpha$, they will be far from each other. This is a

Spatial Selection of Pivots(SSS):
[1] $pivotSet \leftarrow x_1$
[2] $forall s_j \in S do$
[3] $\quad if \forall p \in pivotSet, d(s_j) \geq M\alpha$
[4] $\quad\quad pivotSet \leftarrow pivotSet \cup s_j$
[5] End if
[6] End for

Figure 4.1: **SSS Algorithm**

desirable characteristic for pivots selected by pivot selection techniques[8]. The pivots in SSS are not too far from the rest of the objects in the collection and this means that they are well distributed in space. They are not too far from each other and rest of the objects in the database. This is a desirable property good pivots must have [6, 8].

The algorithm is dynamic and adaptive. The set of pivots adopt itself to the growth of the database. When an object s_{new} is inserted to the database, it is compared against the pivots already selected. The method is efficient, dynamic and adaptive. However, it has a drawback when the distribution of objects is clustered. It selects a large amount of objects as pivots or an inadequate amount of objects as pivots when distribution is clustered or not normally distributed.

Chapter 5

Dynamic HKvp

Kvp based methods outperform other pivot based methods in terms of space complexity while not causing a significant increase on query costs. Furthermore, they reduce the CPU overhead significantly. Pivot based methods like AESA [36] and LAESA [28] store all the distances between pivots and objects. This fact limits the usage of these methods in large databases. Even though pivot-based methods outperform tree-based methods in terms of performance, they use a static pivot set except SSS [6] and this causes several problems when database size grows. First of all, the number of pivots should be increased to perform at a sublinear complexity. Secondly, initial pivots may perform poorly in eliminating newly inserted objects coming from possibly different regions.

SSS [6] is a dynamic method introduced as a LAESA variant that allows insertions of new database objects. It promotes some of the new objects as pivots dynamically. However, it has serious drawbacks that result in poor query performance for clustered or otherwise skewed distributions. SSS is optimized for a symmetrical, balanced distribution and for a specific radius value.

Tree based methods are dynamic in nature. They select new pivots when there are node splits. The split algorithm promotes some objects as representatives. However, tree based methods are more complex to implement and they are more expensive in terms of number of distance computations than pivot based

methods since they use less pivots. To the best of our knowledge, the best of the tree based algorithms is DF-Tree [33] which uses extra global representatives in addition to the pivots of the tree structure. It uses the Omni-HF Algorithm [18] to dynamically select these global representatives when there are objects inserted into the database. We can think of the DF-Tree concept as the Slim-Tree [32] implementation of Omni [18] with global representatives. HKvp structure has the advantage of being able to increase the number of pivots to improve the query performance. Using more pivots is not problematic in some cases like Kvp, since HKvp optimizes the number of pivots to be used. This is performed by a parameter of HKvp called *drop rate* [10]. Even though HKvp has the drop rate parameter, currently it does not have an optimization scheme to determine the parameter effectively. Current implementation takes the drop rate value as a parameter.

From these discussions, we end up with two problems to make HKvp dynamic: drop rate optimization should be satisfied and a dynamic pivot selection algorithm should be adapted to HKvp. In this chapter, we propose methods to estimate drop rate parameter at the query time. We implement the SSS pivot selection algorithm, which is stated to be the best among the others, to HKvp. As stated in Chapter 4, we know that SSS pivot selection technique has a drawback in clustered distributions and in distributions not normally distributed. We propose an alternative to SSS which we call Distribution Based Pivot Promotion(DBPP) to overcome this difficulty. SSS promotes new objects based on a fixed ratio of the maximum possible distance. We propose a new method that takes the distribution of distances into account, not just the maximum possible distance. In this way, we may select the pivots in well distributed amounts even for a clustered distribution or a distribution that is skewed.

5.1 Optimizing HKvp Drop Rate

Rather than requiring an extra parameter, HKvp may be improved by using cost formula of Equation 2.1. Assume that there are p pivots used in the first step,

while there are k pivots overall. Then, there are k-p pivots left to be considered which are eliminated in the first step. If we have a drop rate of α, the number of pivot distances calculated is expressed by $p+(1\text{-}\alpha)\times(k\text{-}p)$.

p= # of pivots processed in the first step
N = number of intervals
fc = failure to cut ratio

[1] $preCost = +\infty$, $cCost = 1$, $lo = 0$, $hi = 1$
[2] While |1-cCost/preCost|>ϵ
[3] $minval = +\infty$
[4] for j=0 to N
[5] $cdrop = lo + j$ *$((hi\text{-}lo)/N)$;
[6] $pivotsUsed = (p + (pivotNumber\text{-}p)*(1\text{-}cdrop))$
[7] $cost = pivotsUsed + databaseSize \times fc^{\,pivotsUsed}$
[8] if $cost<minval$
[9] $minval = cost$, $min_j = j$, $mincdrop = cdrop$
[10] End for
[11] $nlo = lo$, $nhi = hi$
[12] if $min_j>1$
[13] $nlo = lo+(min_j\text{-}1)*((hi\text{-}lo)/N)$
[14] End if
[15] if $min_j<N\text{-}1$
[16] $nhi = lo+(min_j+1)*((hi\text{-}lo)/N)$
[17] End if
[18] $lo=nlo$; $hi=nhi$, $preCost=cCost$, $cCost=minval$;
[19] End While
[20] The drop rate α = mincdrop

Figure 5.1: **Drop Rate Estimator Algorithm**

If we consider the cost Equation 2.1, then the cost can be estimated by Equation 5.1.

$$Cost(q,r) = p + (1-\alpha)\times(k-p) + nfc^{\,(p+(1-\alpha)\times(k-p))} \tag{5.1}$$

This formula may be optimized for each query radius value. This is performed by the *drop rate estimator algorithm* shown in Figure 5.1.

In this section, we will propose several drop rate optimization methods that differ in the way they estimate the value of fc. These methods work dynamically for each query. That is, they determine the correct drop rate value while the query is being processed.

5.1.1 PCAIPD

Pivot Cut Approximation Based on Inter Pivot Distances(PCAIPD) is the first method we will consider for drop rate detection. It has two different implementations PCAIPD-Avg and PCAIPD-Root. PCAIPD-Avg and PCAIPD-Root optimize the drop rate by looking at the elimination power of pivots chosen at first phase. We will first discuss PCAIPD-Avg. PCAIPD-Avg averages the sum of probabilities of a pivot failing to eliminate another pivot from the rest of the pivot set. Assume that we have k pivots. First pivot eliminates x_1 pivots from these k pivots, then there are k-x_1 pivots left. The second pivot eliminates x_2 pivots from pivot set and k-x_1-x_2 is left after we process the second pivot. The process continues until the number of pivots to process is 0. The failure to cut probability of the first pivot is $\frac{k-x_1-1}{k-1}$ where the failure to cut probability of the second pivot is $\frac{k-x_1-x_2-1}{k-x_1-1}$. For the i^{th} pivot this fraction is equal to $\frac{k-x_1-x_2-...-x_i-1}{k-x_1-...-x_{i_{-1}}-1}$ Assuming that p pivots are selected at first phase, then the failure to cut probability for PCAIPD-Avg is expressed by equation 5.2 by taking the average failure to cut values.

$$fc = 1 - \frac{\sum_{i=1}^{p}(\frac{x_i}{k-1-\sum_{j=1}^{i-1}x_j})}{p} \tag{5.2}$$

The same failure to cut probabilities apply for PCAIPD-Root. However, this time we calculate the geometric mean rather than the arithmetic mean of the fc values of each pivot. In other words, we take the p^{th} root of multiplication of the failure to cut probabilities which is equal to $\frac{k-x_1-1}{k-1} \times \frac{k-x_1-x_2-1}{k-x_1-1} \times ... \times \frac{k-x_1-x_2-...-x_p-1}{k-x_1-...-x_{p_{-1}}-1}$. From this multiplication, we arrive at equation 5.3 for the

failure to cut probability.

$$fc = \sqrt[p]{1 - \frac{\sum_{i=1}^{p} x_i}{k-1}} \tag{5.3}$$

PCAIPD generally gives good results except for clustered distributions. In higher dimensions, we also see that PCAIPD gets worse in performance when compared to current drop rate detection scheme which we call as optimal drop rate. Even though PCAIPD does not give the best results among the drop rate detection algorithms, it is observed to be especially successful for uniform data. PCAIPD does not use any extra information than the algorithm itself. It is a heuristic algorithm and it only uses the cost formula of Equation 5.1 with the set of pivots to process after first step of the HKvp algorithm.

5.1.2 PCAGD

Pivot Cut Approximation Based on General Distribution(PCAGD), samples $l \times n$ distance computations between n database objects where l is a constant to determine how many distance computations will be considered. From this set of distances, the general distribution of the database is estimated. Alternatively, the distance computations acquired during initial pivot selection can also be used for distance distribution approximation. Our experiments showed that this method yields a close approximation. As a data structure, an array of m distribution intervals is used to approximate the distance probability function. The failure to cut probability is calculated assuming that the query object is selected to be in the middle of each interval symbolized as $precision*i+precision/2$ for each $i \in m$. The failure to cut probability is expressed by Equation 5.4 which takes the weighted sum of F(d_i+r)-F(d_i-r), the failure to cut value for a pivot p_i at a distance d_i to the query object. So we assume that the possibility of having a pivot at a distance d_i to the query object is equal to $f(d_i)$, the probability of any object to be at a distance d_i to the query object.

$$fc = \sum_{i=0}^{m} f(d_i) \times (F(d_i + r) - F(d_i - r)) \tag{5.4}$$

Figure 5.2 shows the area where an object can not be eliminated for a range query with query object q, pivot p and radius r.

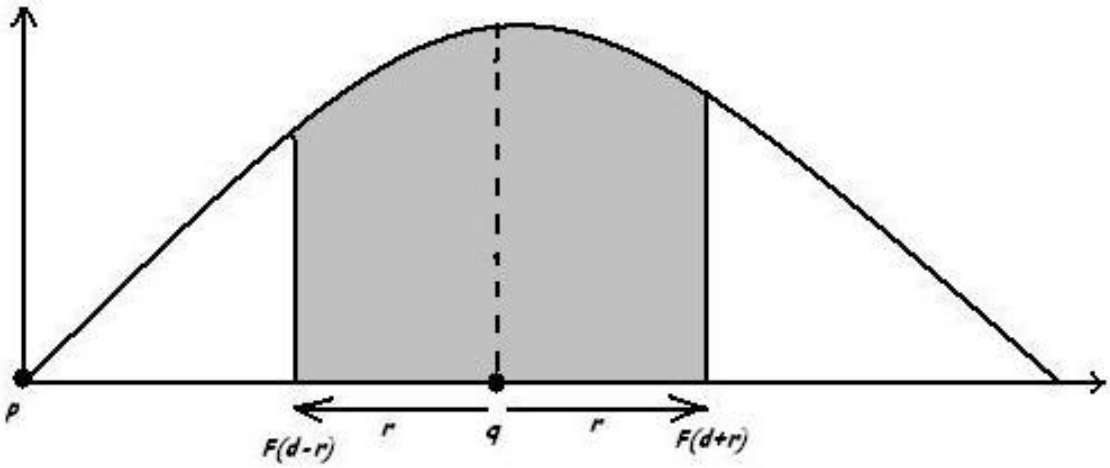

Figure 5.2: **Failure to Cut Region**

PCAGD is very successful in detecting a drop rate ratio to minimize the number of distance computations. This is because it uses more information than PCAIPD variants.

5.1.3 PCAPQPD

Pivot Cut Approximation Based on Query Pivot Distances(PCAQPD) operates similar to PCAGD. PCAGD assumes that pivots come from the general distribution, and weights the cut probabilities over the whole range of distances. PCAPQPD uses the actual set of pivots $PP \subseteq P$ that are processed in the first phase to determine the average fc that will be in affect during the processing of the query. Rather than being a static calculation as PCAGD, PCAPQPD estimates fc for each query separately. Equation 5.5 summarizes the failure to cut probability for PCAQPD.

$$fc = \frac{\sum f(d_{pq}) \times (F(d_{pq}+r) - F(d_{pq}-r))}{\sum f(d_{pq})}, \forall p \in PP \tag{5.5}$$

The number of pivots $PP \subseteq P$ evaluated in first phase of PCAQPD HKvp is adequate to represent the whole behavior of data set.

5.1.4 PCAPP

Pivot Cut Approximation Based on Pivot Performance is the only method that has a calibration phase among the proposed methods. It starts out by processing objects from database under a drop rate value of 1. After processing these objects, the algorithm determines a drop rate value with respect to the failure to cut probability of the processed objects. Assuming that there are 10000 objects in the database and 1000 objects are selected for calibration phase, then failure to cut probability fc is equal to $\frac{number\ of\ eliminated\ objects}{1000}$. Since the range query starts with a drop rate of 1, we only have the failure to cut probability for first phase pivots $PP \in P$. Thus, assuming the cardinality of PP is p, failure to cut probability fc for a pivot is approximated as $\sqrt[p]{\frac{number\ of\ eliminated\ objects}{1000}}$. If we write this mathematically, Equation 5.6 arises.

$$fc = \sqrt[p]{\frac{number\ of\ eliminated\ objects}{number\ of\ objects\ used\ in\ calibration}} \tag{5.6}$$

PCAPP is very successful for determining the drop rate value. PCAPP detects the drop rate on the fly. Therefore, one may expect that it would work better than PCAGD. It is not as sharp as PCAGD in some situations.

5.2 Distribution Based Pivot Promotion

Spatial Selection of Sparse Pivots(SSS)[6] outperforms the other pivot selection techniques. It suggests a strategy which automatically detects the number of pivots to be used. It adapts itself to the dimensionality of the database. It allows object insertions unlike the previously defined pivot selection mechanisms. These properties of SSS makes it ubiquitous for our HKvp purposes. However, it is

not successful at clustered distributions and skewed distributions. This is due to the pivot selection mechanism which selects a pivot if and only if its distance to any pivot in the current set of pivots is equal to or larger than $M\alpha$ where α is a constant and M is the maximum distance between two objects from the data set. In clustered distributions, there may be more than one peak and data is not well distributed. Therefore, the number of pivots selected may deviate from the projected optimal value.

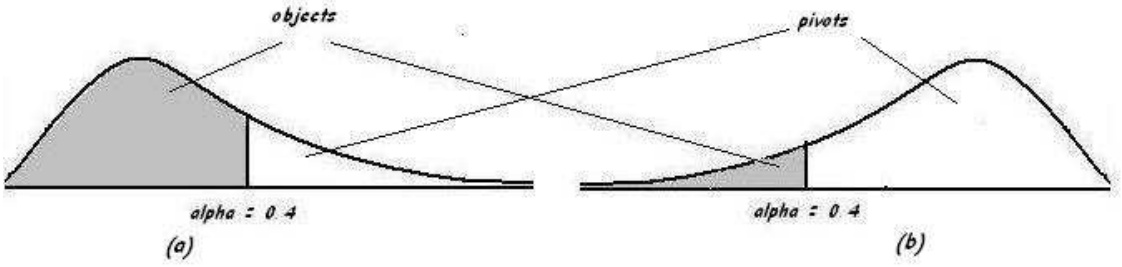

Figure 5.3: **SSS Pivot Selection**

In Figure 5.3, we see the both situations that may apply. In the Figure 5.3-a, we see the situation when number of pivots will be less than optimal since only few objects can be far than the selected distance. On the other hand, Figure 5.3-b shows the situation when there will be more pivots than the optimal since most of the objects can beat the threshold distance. We have implemented SSS with α value 0.4 since the authors in [6] suggest 0.4 for α value. This is an empirical result arose from SSS experiments [6]. Distribution Based Pivot Promotion(DBPP) outperforms SSS while solving this problem by using a percentage of the distance distribution instead of the maximum of the two distances M.

DBPP uses a threshold r_t where $\mathrm{F}(r_t) = \alpha$. If an object is located at a distance larger than this threshold to its closest pivot, it is promoted as a pivot. Unlike SSS, DBPP has the ability to perform well on clustered distributions.

Chapter 6

Performance Results

Our methods have been tested with several data collections in different situations. First, we used synthetic datasets of random points to understand which drop rate optimization technique is better. Among these algorithms PCAPP, PCAQPD and PCAGD give the best results. Second, we compared our DBPP based algorithm versus SSS for google data and synthetic datasets. Finally, we perform some comparisons between the most popular disk based method DF-Tree in terms of adaptation, insertion and scaling tests with our Dynamic HKvp structure. The results show that Dynamic HKvp structure outperforms the DF-Tree and SSS.

6.1 Overall Comparison of Drop Rate Detection Techniques

We proposed several drop rate detection algorithms up to now. We show that some of these methods perform very close to the optimal value. Among these algorithms PCAPP, PCAQPD and PCAGD give the best results as seen in Figures 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 6.10, 6.11, 6.12, 6.13, 6.14.

We have run experiments for a data set of 5K objects under varying pivot limits and dimensions to see how much the drop rate detection algorithm effects the costs and pivot selection. Clustered distributions in experiments use 500 clusters and cluster radius of 0.1.

In order to evaluate the correctness of our methods, we have found the optimal drop rate by using the steepest descent method. We will optimize the drop rate based on the performance of the query after it has been finished. This means that the same query will be executed multiple times to find the optimal drop rate for a particular query. We will call this optimal drop rate as DR_OPT in our Figures. In order to compare our methods against the conventional pivot based methods like Spaghettis, SSS and OMNI, we will use the symbol DR=0 to show the fact that all the pivots are used without dropping any.

Unfortunately PCAIPD variants were not successful to detect the optimal drop rate. A reason of this result may be that the pivots selected in first phase of the HKvp Algorithm does not represent the overall elimination behavior of all pivots. We see that PCAPP, PCAGD and PCAQPD perform at a similar performance. For the remaining experiments, we will use the PCAPP to compare our scheme with other methods in literature.

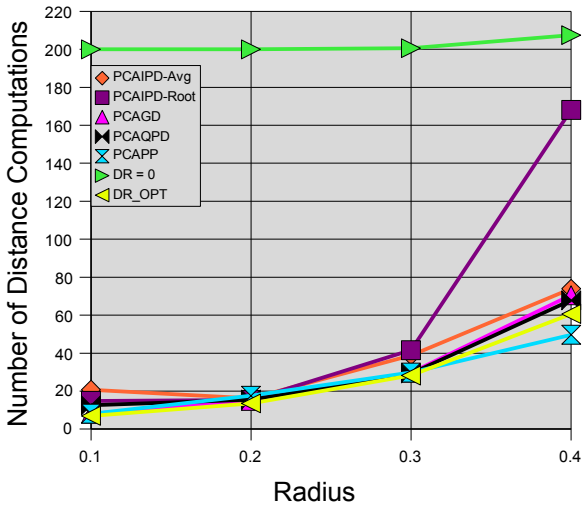

Figure 6.1: **Overall Comparison 5000 Uniform Vectors Dimension=10 Pivot Limit=200**

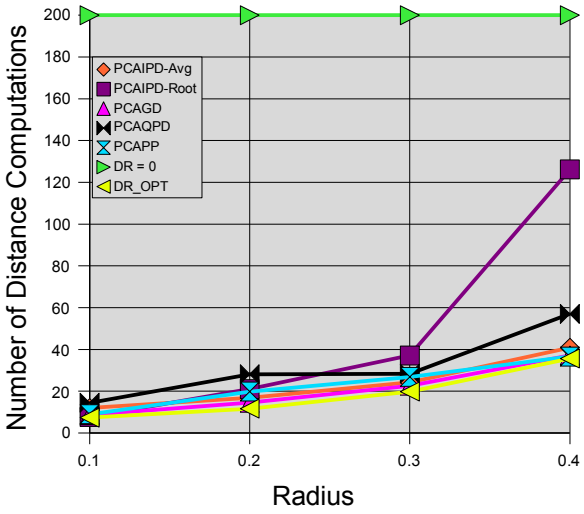

Figure 6.2: **Overall Comparison 5000 Uniform Vectors Dimension=30 Pivot Limit=200**

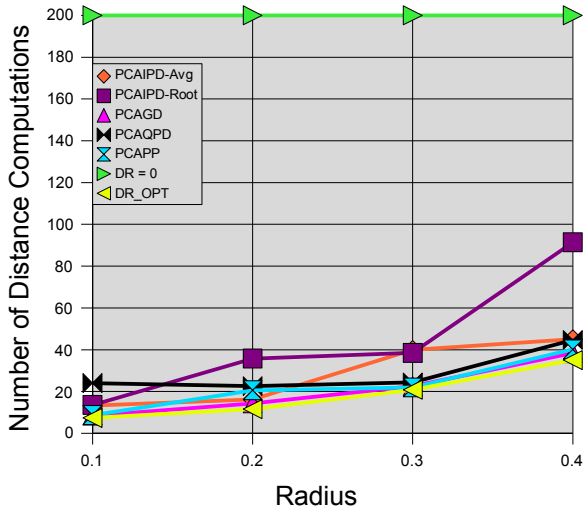

Figure 6.3: **Overall Comparison 5000 Uniform Vectors Dimension=50 Pivot Limit=200**

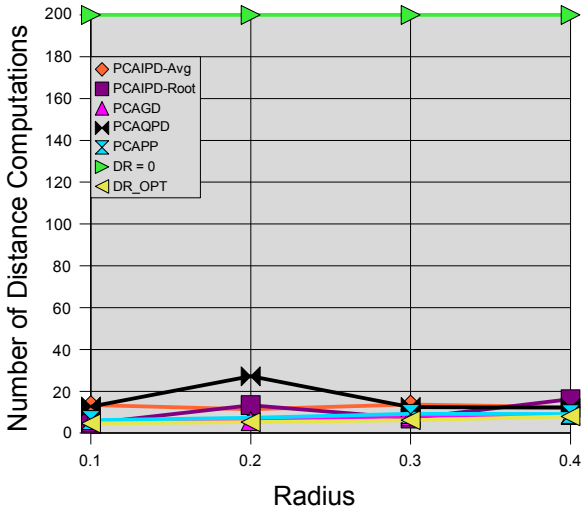

Figure 6.4: **Overall Comparison 5000 Clustered Vectors Dimension=10 Pivot Limit=200**

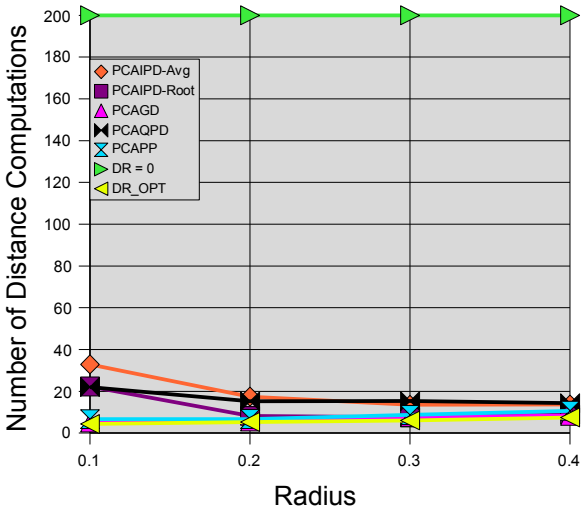

Figure 6.5: **Overall Comparison 5000 Clustered Vectors Dimension=30 Pivot Limit=200**

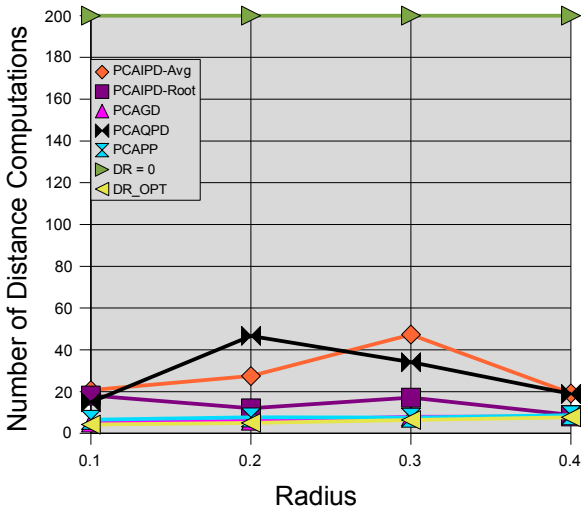

Figure 6.6: **Overall Comparison 5000 Clustered Vectors Dimension=50 Pivot Limit=200**

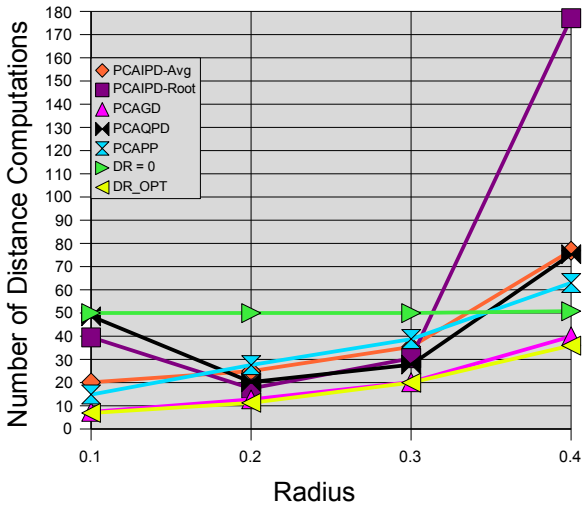

Figure 6.7: **Overall Comparison**
5000 Uniform Vectors
Dimension=30
Pivot Limit=50

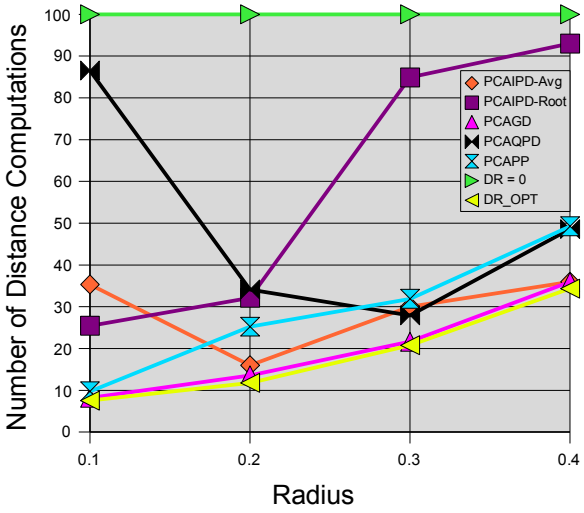

Figure 6.8: **Overall Comparison**
5000 Uniform Vectors
Dimension=30
Pivot Limit=100

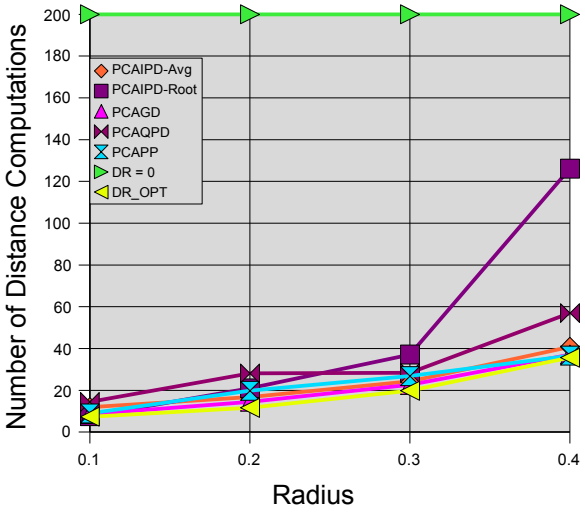

Figure 6.9: **Overall Comparison**
5000 Uniform Vectors
Dimension=30
Pivot Limit=200

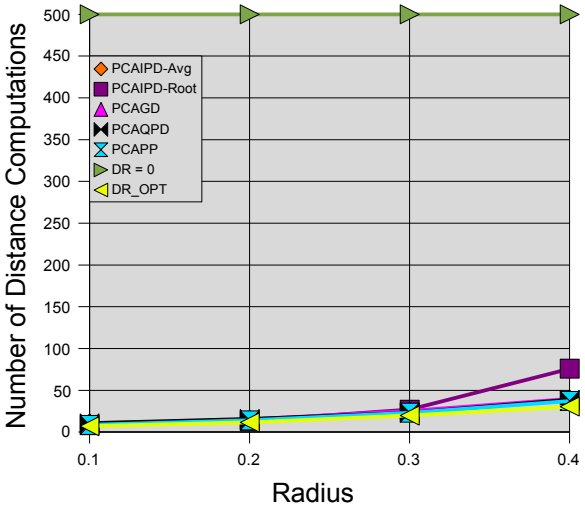

Figure 6.10: **Overall Comparison**
5000 Uniform Vectors
Dimension=30
Pivot Limit=500

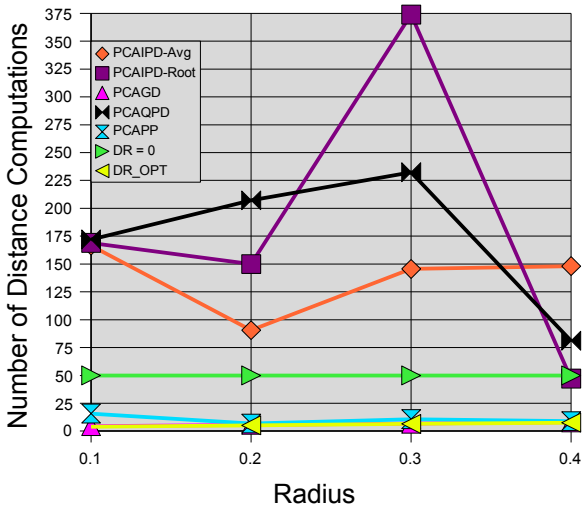

Figure 6.11: **Overall Comparison 5000 Clustered Vectors Dimension=30 Pivot Limit=50**

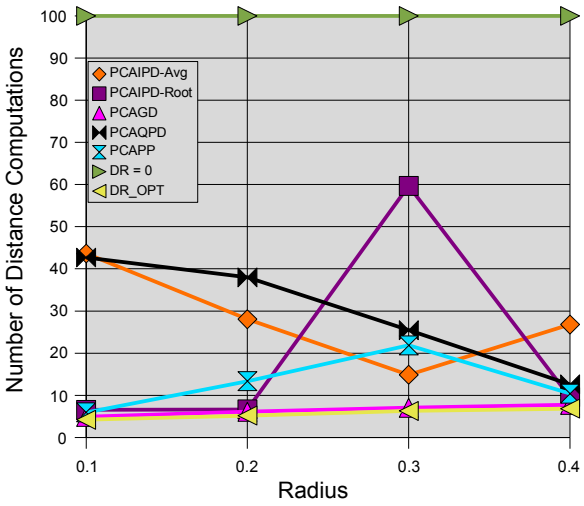

Figure 6.12: **Overall Comparison 5000 Clustered Vectors Dimension=30 Pivot Limit=100**

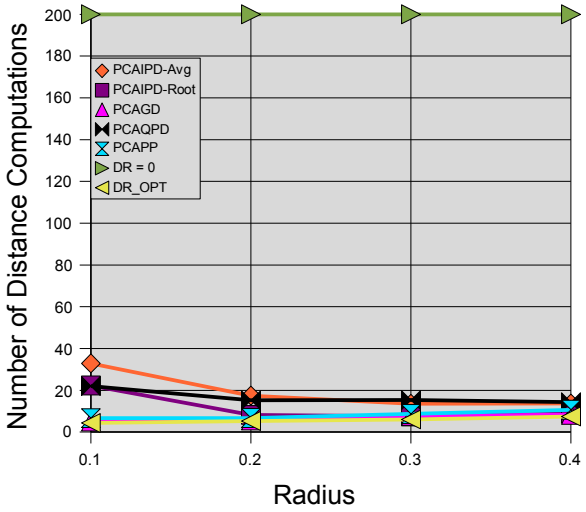

Figure 6.13: **Overall Comparison 5000 Clustered Vectors Dimension=30 Pivot Limit=200**

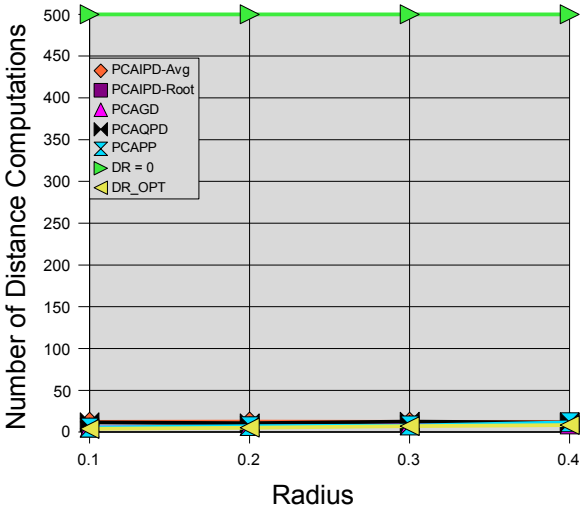

Figure 6.14: **Overall Comparison 5000 Clustered Vectors Dimension=30 Pivot Limit=500**

6.2 DBPP versus SSS

In our experiments, we used the google data set. Google data set [2] consists of a set of web pages obtained by Google Inc [2]. The pages are converted into term vectors and cosine distance is used. The distribution of this dataset is skewed to the right, meaning that we expect the SSS to select too many pivots as in Figure 6.17-b. We have experimented DBPP versus SSS on a 1K and 5K of web data. The experiments in Figure 6.15 show the performance of the DBPP over SSS. We used PCAPP and PCAGD in our DBPP experiments. The distribution of this dataset is skewed to the right, meaning that we expect the SSS to select too many pivots as in Figure 6.17-b. Indeed, there were 970 pivots for SSS and 74 pivots for DBPP for the dataset with 1000 objects. For the 5K dataset, these values were 4758 for SSS and 151 for DBPP. We see from Figure 6.15 that for both the 1K and 5K data, DBPP outperforms SSS technique. DBPP-Normal in Figures mean drop rate is equal to the optimal value for DBPP.

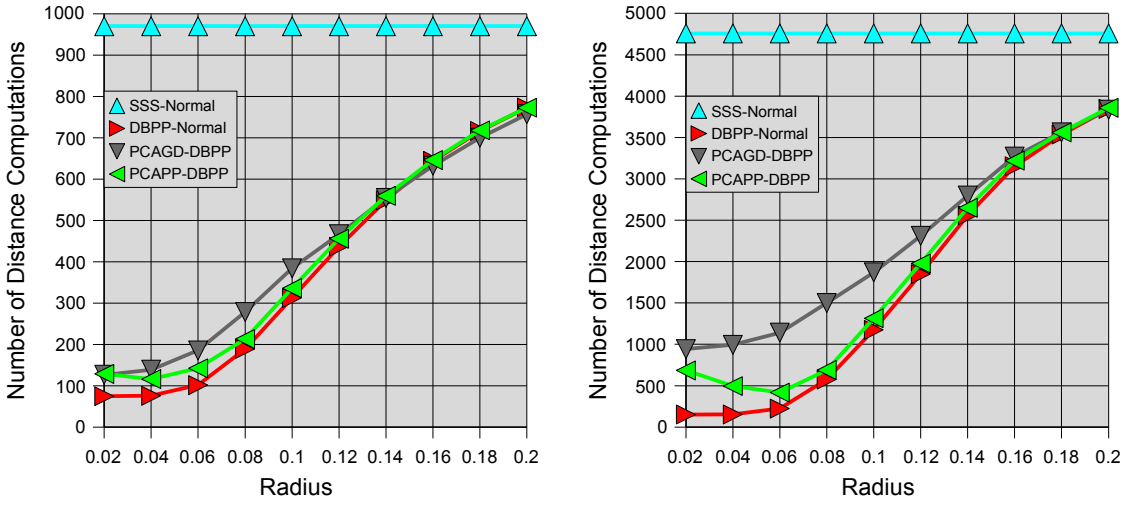

Figure 6.15: **DBPP versus SSS on 1K(left) and 5K(right) Data, $\alpha = 0.4$**

Our DBPP approach outperforms SSS not only in uniform and clustered distributions but also in skewed distributions like google data. Since we know that for higher values of alpha the number of pivots is limited, it is not meaningful to use alpha values larger than a value. SSS suggests 0.4 for its optimum value. Drop optimized HKvp variants are not dependent on the low values of alpha like

SSS because of the drop rate mechanism cuts the number of extra pivots from computational costs. However, if we select a high alpha value such that HKvp uses less pivots than it should use, we may face with higher costs.

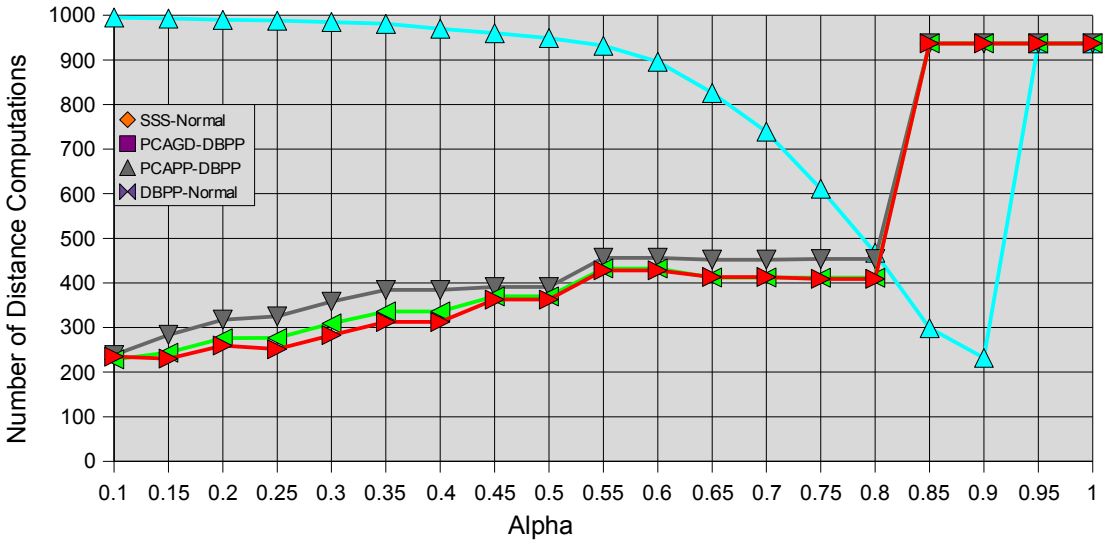

Figure 6.16: **DBPP versus SSS on 1K Data, query radius = 0.1**

Figure 6.16 shows DBPP and SSS costs for varying alpha values. We see that after the value of alpha reaches 0.8, HKvp performance reduces because the number of pivots selected is less than expected. We see that, SSS reaches an optimum cost only after this value. This also shows that SSS algorithm is not stable and sharp changes in alpha value may cause drastic results. DBPP shows the best result at an alpha value of 0.1. However, this may be misleading for one to determine whether it is suitable to use that alpha value. There is a trade-off between insertion cost and query cost in DBPP. If we select higher number of pivots, we end up with a higher insertion cost. This relationship between insertion cost and query performance will be investigated later.

Another point to mention is the behavior of SSS and DBPP when dimension changes. We expect the computational costs to increase when the dimension increases because the data distribution gets skewed. The increase in dimensionality means the increase in distances of objects to each other. We see from Figure 6.17 that this idea is true for higher dimensions. An interesting point is that even a dimension of 50 is enough for SSS to deteriorate. We used a clustered data of 5K

with 500 pivots and 50 for pivot limit. Alpha, cluster radius and query radius are selected as 0.1. The results are much more better than SSS.

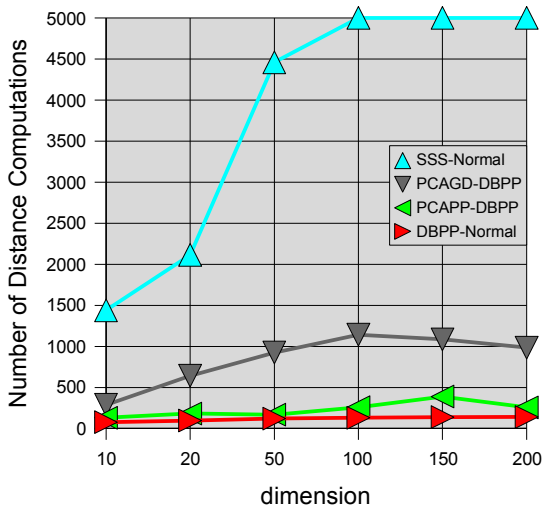

Figure 6.17: **DBPP versus SSS on 5K Clustered Data, query radius = 0.1, alpha = 0.1, pivot limit = 50**

As we previously mentioned, SSS is optimized for specific radius values. As we see in Figures 6.18, 6.19, 6.20 PCAGD-DBPP and DBPP-Normal outperform SSS. We see that SSS performs at very high costs even for lower dimensions and lower radii.

All in all, SSS has fundamental problems that results in poor query performance for clustered or otherwise skewed distributions. Real datasets have often been observed to show such characteristics. We showed that SSS has been optimized to work for a symmetrical, balanced distribution and for a specific radius value. We proposed our alternative approach DBPP which solves all of these problems.

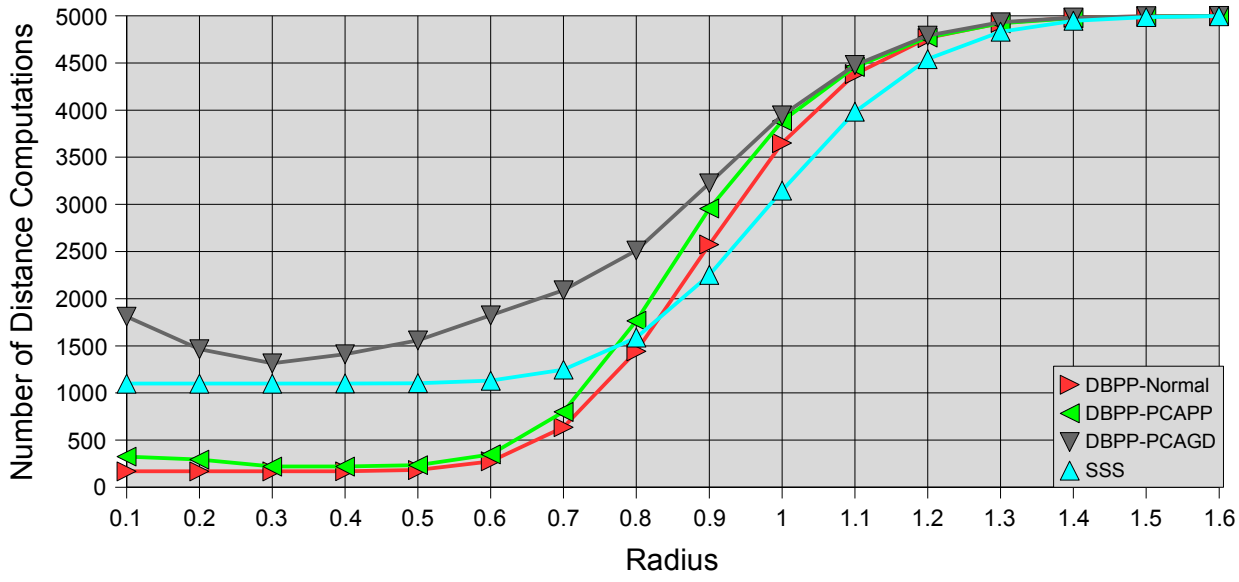

Figure 6.18: **DBPP versus SSS on 5K Uniform Data, dimension= 20, pivot limit = 50, alpha = 0.4**

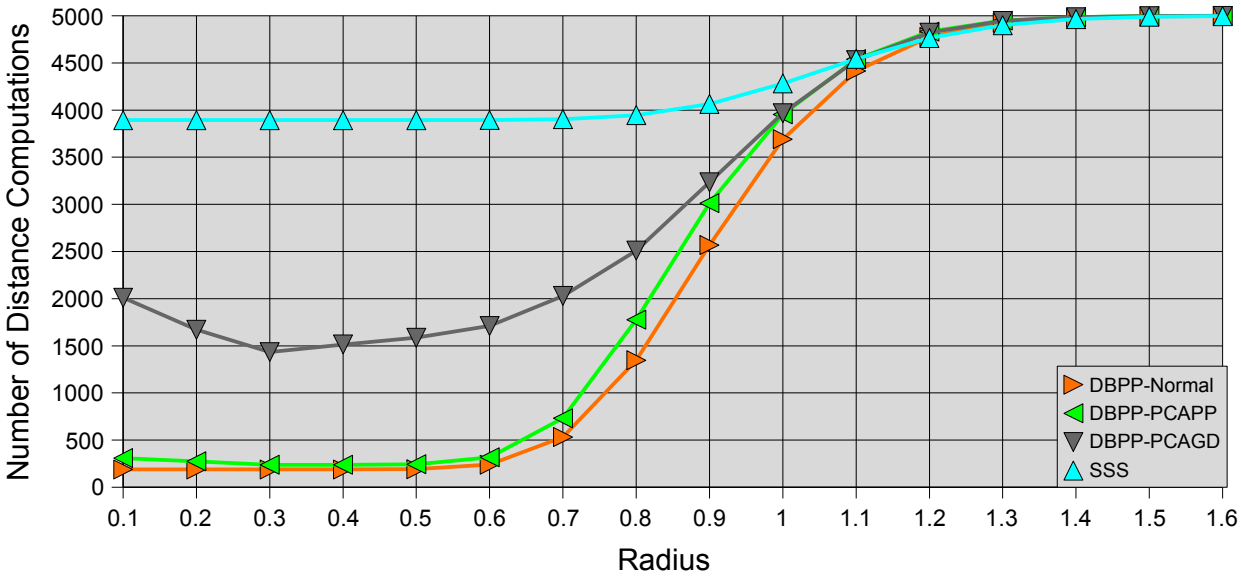

Figure 6.19: **DBPP versus SSS on 5K Uniform Data, dimension= 30, pivot limit = 50, alpha = 0.4**

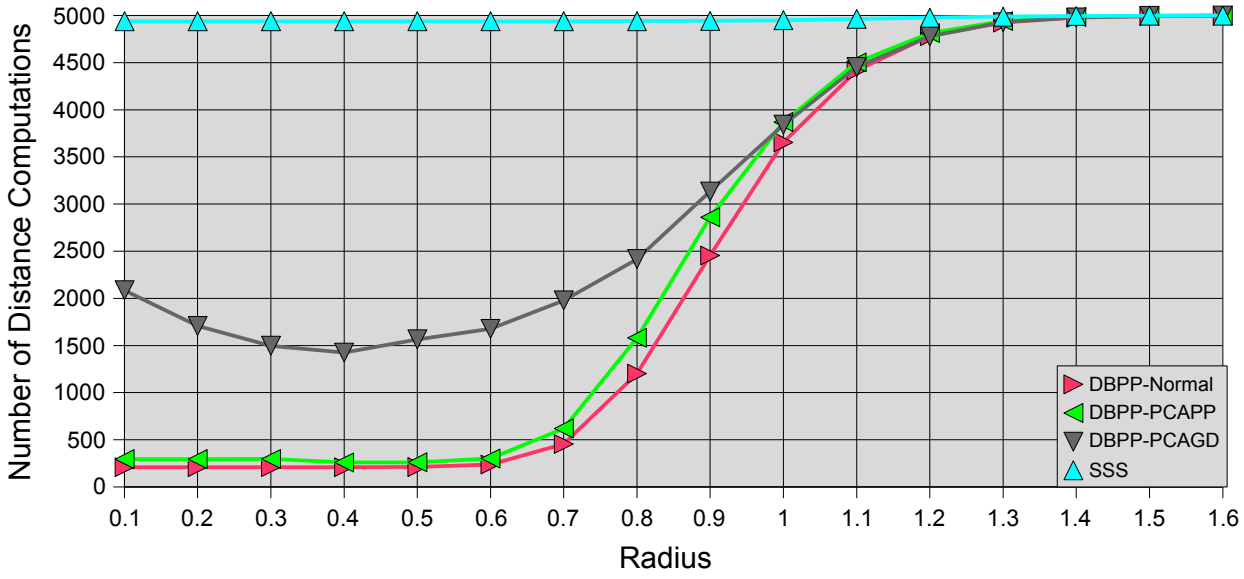

Figure 6.20: **DBPP versus SSS on 5K Uniform Data, dimension= 40, pivot limit = 50, alpha = 0.4**

6.3 DF-Tree versus PCAPP-DBPP-HKvp

We aim to use HKvp for large data sets, especially stored in secondary storage. To the best of our knowledge, DF-Tree [33] is the most effective dynamic technique used in large data sets. It is a Slim Tree [32] based algorithm which allows global representatives. To select the global representatives, the DF-Tree uses Omni-HF Algorithm [18]. In this section, we perform some tests to compare HKvp and DF-Tree.

The drawback of the general HKvp algorithm is that it does not have the ability to determine the drop rate. By PCAPP and PCAGD, we solved the drop rate detection problem. This was an improvement about the performance of the HKvp algorithm. After this procedure, we have implemented the state of the art pivot selection technique SSS to HKvp. We proposed and implemented an alternative pivot selection technique on HKvp called DBPP which outperforms SSS over real data. We tested our HKvp structure versus DF-Tree structure under scaling, adaptation and insertion tests. The HKvp capabilities were based on DBPP and PCAPP in these experiments. Scaling test evaluates the query performance of the algorithms, while adaptation test tries to determine whether the algorithm may select pivots with respect to the needs of the frequently changing dataset. Insertion test is performed to determine the insertion costs of newly arriving objects.

6.3.1 Query Performance

The critical factor that determines the success of a method is the query cost. The purpose of all metric space indexing methods are to decrease the query costs while not causing any other costs in terms of construction and insertion. When performing scaling tests, we followed three strategies. We first started with varying the dimension parameter while other parameters are fixed as in Figure 6.21. When the dimension increases, the distances between objects are assumed to increase. This causes the number of objects below a specific radius

to decrease. We see that HKvp outperforms DF-Tree in all dimensions. The gap widens especially for higher values of the dimensions.

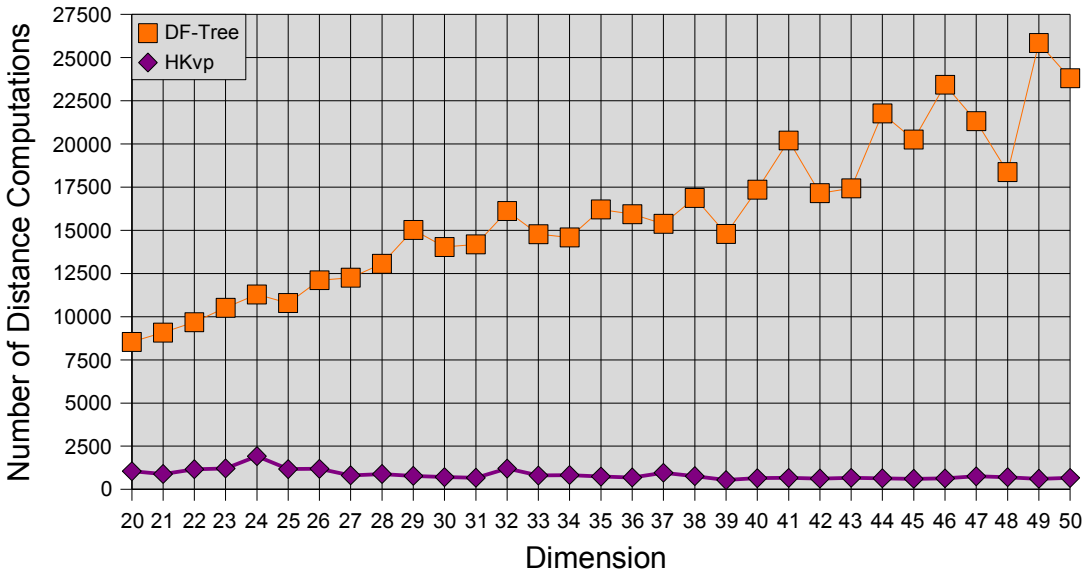

Figure 6.21: **Query Performance for Varying Dimension, Data Size=50K, Radius=0.3, Pivot Limit = 500**

The second strategy to compare query performance was varying the query radius while other parameters are constant. The important thing to consider is the determination of the radius value and what percentage of objects would probably be selected by that query. Selecting the radius such that half of the database or more than half of the database is retrieved from a data set makes the metric space approaches meaningless. In Figure 6.22, we see that HKvp outperforms DF-Tree in all radius values but it shows the best performance for lower, more common radius values.

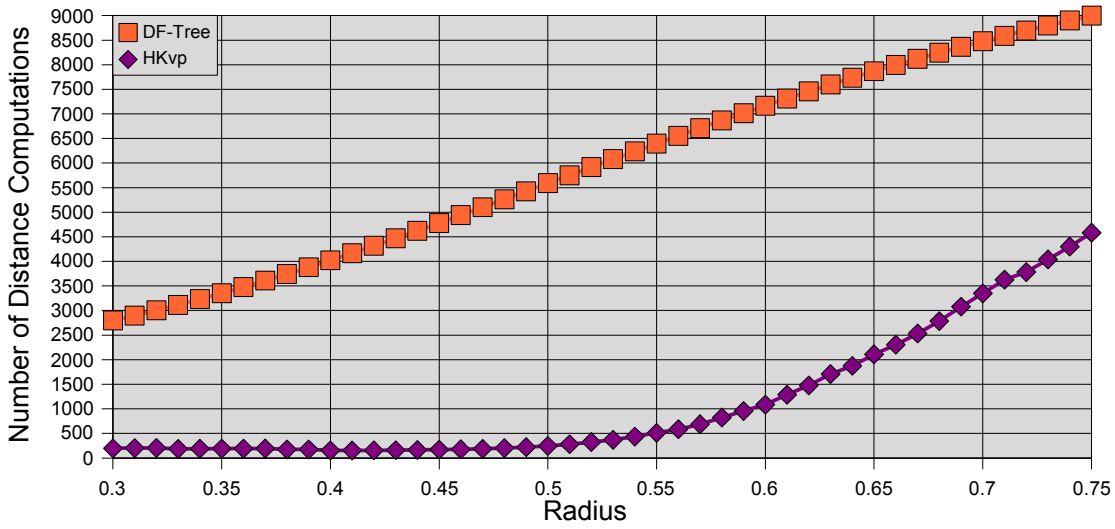

Figure 6.22: **Query Performance for Varying Radius, Data Size=10K, Dimension=30, Pivot Limit = 500**

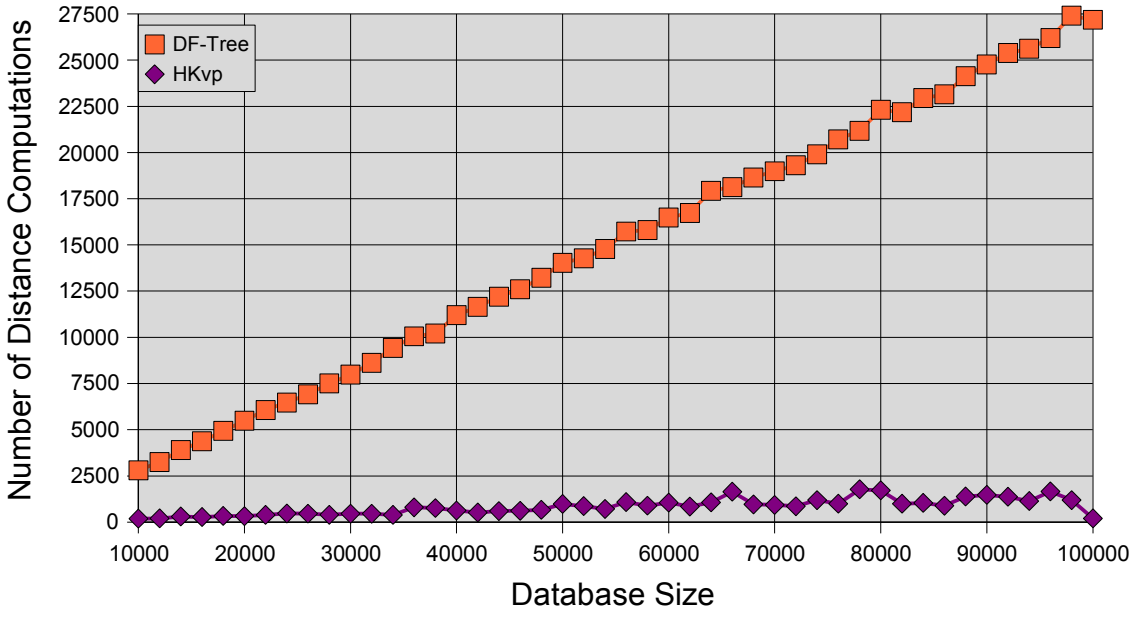

Figure 6.23: **Scaling Test for Varying Data Size, Radius=0.3, Dimension=30, Pivot Limit = 500**

The last strategy we proposed was scaling comparisons. We increased the database size while other parameters are fixed as seen in Figure 6.23. Costs for DF-Tree increases linearly, while HKvp is not affected. It may be strange that at some points costs of HKvp decreases while database size increases. This may be because of increasing number of pivots.

HKvp shows better results because HKvp may drop whenever there are more pivots than needed or use them when all the pivots are needed. DF-Tree chooses global representatives only after a bulk insertion and generally selects the number of global representatives less than or equal to intrinsic dimension of data set. This causes DF-Tree to choose less pivots and perform worse than HKvp.

All in all, we understand from the scaling tests in Figures 6.21, 6.22 and 6.23 that HKvp outperforms the DF-Tree structure. Figures 6.21 and 6.23 are strange from a point. While dimension or data size increases, there are sharp inclines and declines in the graphs. This may be because of the adaptation of the structure to the growing database difficulty through data size and dimension. Peaks are larger in DF-Tree plots because DF-Tree modifies the pivot set after a group of insertions unlike HKvp. HKvp decides if an object is better to be a pivot in object insertion. We see from Figure 6.22 that the gap widens between HKvp and DF-Tree when the query radius r of range query increases. This is due to the increase in the difficulty of the query with the increase of radius.

6.3.2 Adaption Tests

Adaptation tests are the tests to evaluate how much the metric access method adapts to newly inserted objects. For these tests, we arranged two regions far from each other. These regions are equal in size and they do not intersect. The query objects are selected from one of these regions. We call the two regions as Region A and Region B. Region A consists of the vectors where each dimension can take values in the range [0,1]. For Region B, this range is [-1,0]. The tag DF-TreeAB in Figures 6.24, 6.25 and 6.26 means we insert the first half of the dataset from Region A and the rest from Region B. The tag DF-TreeBA means

we insert the first half of the dataset from Region B and the rest from Region A. We selected the query objects from Region A for our experiments. This way, we can see whether the index structures can adopt to the changes in database population.

For the adaptability tests, we can say that HKvp is more adaptable to radius changes than DF-Tree by looking at Figure 6.25 while DF-Tree performs well in terms of adaptability in varying data sizes as seen in Figure 6.24. Both algorithms fail to adapt for different dimensions as seen in Figure 6.26. However, DF-Tree shows sharper declines and inclines and it has much more higher costs than HKvp.

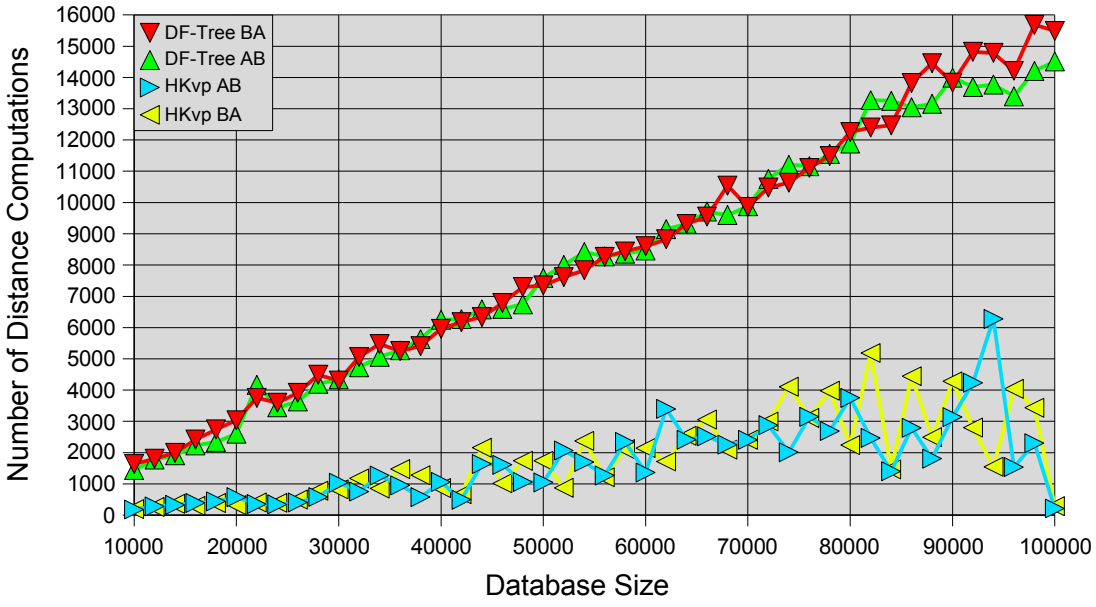

Figure 6.24: **Adaptation Test for Varying Data Size, Dimension=30, Radius=0.3, Pivot Limit = 50**

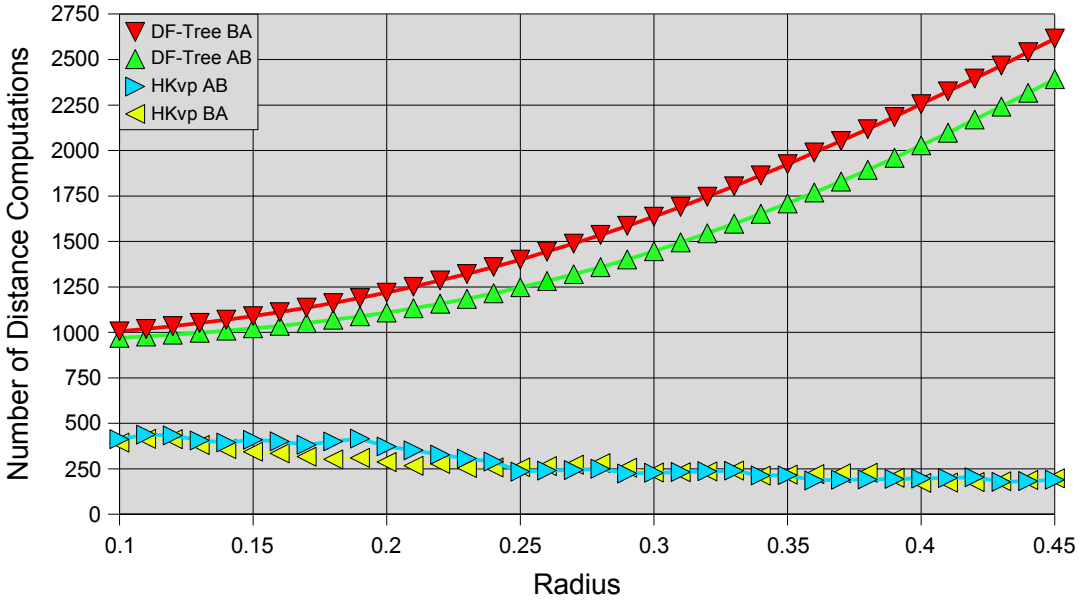

Figure 6.25: **Adaptation Test for Varying Radius, Data Size=10K, Dimension=30, Pivot Limit = 50**

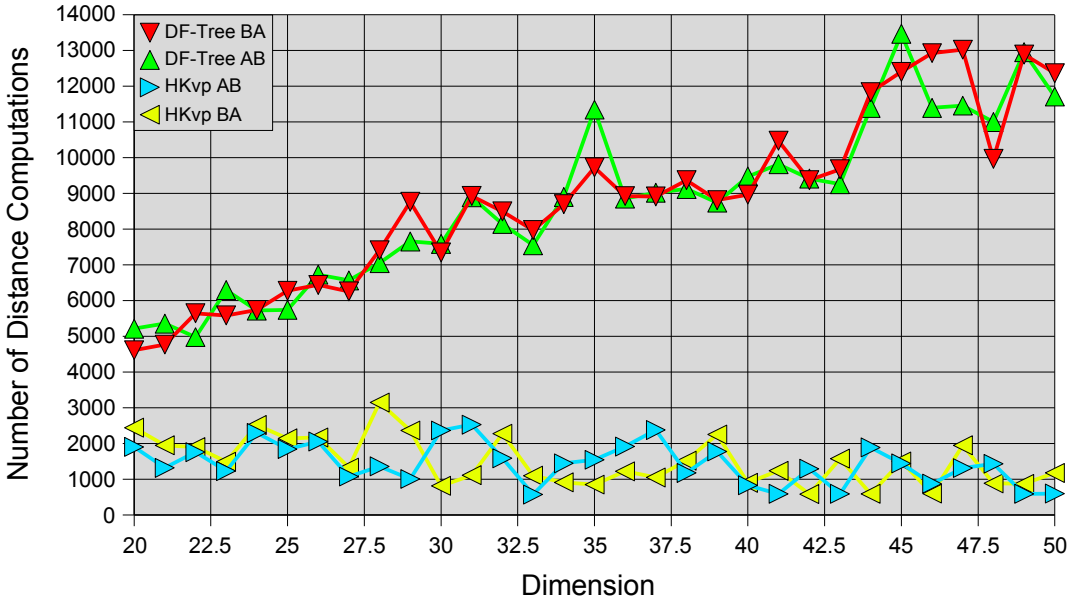

Figure 6.26: **Adaptation Test for Varying Dimension, Data Size=50K, Radius=0.3, Pivot Limit = 50**

6.3.3 Insertion Tests

An insertion into the DF-tree may cause splits in different levels of the tree. We will show that the insertion cost follows a linear trend. For HKvp, assuming there are k pivots at the time of the insertion, a regular insertion will cause k distance computations. However, sometimes a new object is promoted as a pivot, forcing all the existing objects to compute their distances to this new pivot. For such cases, the insertion cost is n.

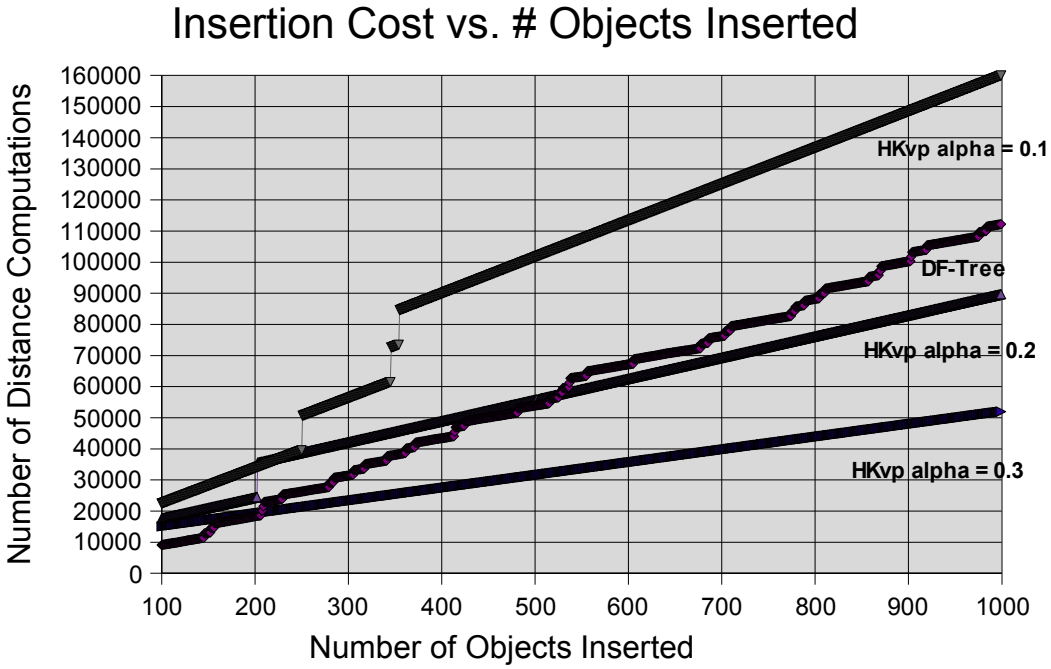

Figure 6.27: **Insertion Test for 1000 objects, Dimension = 10, Data Size = 10K, radius = 0.3, pivot limit = 50**

Figure 6.27 shows the insertion costs to a database of size 10000. We have argued that low values of alpha causes more pivots to be selected, resulting in higher insertion costs. The jumps in the figure occur when a new pivot is promoted. Having more pivots is bad for insertion costs, however it will decrease the query costs. So,there is a trade-off between insertion cost and query cost. Figure 6.28 shows this relationship between insertion cost and query cost. The

single point shows the DF-Tree values of query and insertion costs. For HKvp, we can generate different options by varying the alpha values. We see that with a selection of alpha starting a little above from 0.1, HKvp starts to outperform DF-Tree in terms of both query and insertion costs.

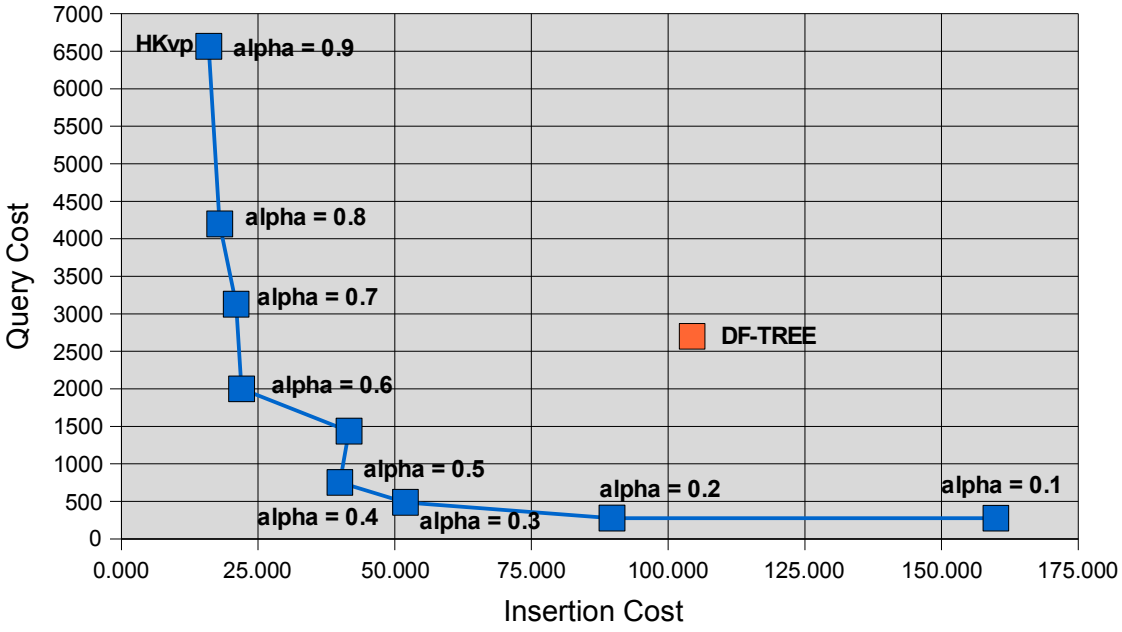

Figure 6.28: **Query Performance vs. Insertion Performance for 1000 objects, Dimension = 10, Data Size = 10K, radius = 0.3, pivot limit = 50**

Chapter 7

Conclusions

In this thesis, we have presented a number of improvements to the prevailing pivot based HKvp structure. Our main contribution is the algorithms we propose to detect drop rate value on the fly. We proposed five different strategies PCAIPD-Avg, PCAIPD-Root, PCAGD, PCAPP and PCAQPD for automatic detection of the drop rate. PCAIPD-Avg uses the arithmetic mean of the failure to cut probabilities and PCAIPD-Root uses the geometric mean of the failure to cut probabilities. PCAGD and PCAQPD are based on the general distribution of the data. PCAPP is different from these algorithms because it has a calibration phase. It determines the failure to cut probability after this calibration phase. Best results have been obtained with PCAPP, PCAGD and PCAQPD gave the best results, therefore we constructed our work on PCAPP and PCAGD.

After successfully determining the drop rate value and optimizing the query costs, we searched for a pivot promotion mechanism to implement our structure on frequently changing dynamic data sets. SSS is the only algorithm that promotes some of the new objects as pivots. Tree splitting algorithms also promote some objects as pivots, however they work on a subset of the objects, and they partition the space into two subsets. This is a very different situation than index structures based on global pivots. SSS has a parameter that determines the rate of the promotions, but this parameter was optimized for a particular query radius value, causing it to perform worse for other radius values. SSS was also observed to

perform poorly when the distance distribution of the dataset is skewed, which is a very common property of real datasets. Depending on the direction of the skewness, it either selects too many or to few pivots. Since SSS does not have the drop rate parameter, it computes the distances between a query object and all the pivots. This results in very high query costs when it selects too many pivots. We proposed a distribution-sensitive method DBPP for deciding when to promote a new object as a pivot. Our tests show that our structure based on DBPP outperforms SSS consistently.

We also compared our scheme with the DF-Tree, which shows the best performance among the secondary storage structures to the best of our knowledge. We show that our method significantly outperforms the DF-Tree in terms of query costs when they use the same insertion cost, and our method has a lower insertion cost when it has similar query performance.

Overall we feel that the techniques and concepts presented in this thesis provide significant improvement to our understanding of dynamic similarity search algorithms. Our method was based on the evaluation of inserted objects as new pivot candidates. For future research projects, the possibility of promoting query objects should also be investigated. Once a query is executed, we already have information about its distances to the existing pivots and some of the database objects. This information can be used to evaluate it as a pivot candidate against the existing pivots. If selected as a pivot, the object distances can be used to significantly decrease the cost of pivot creation.

Bibliography

[1] "GBDI Arboretum Metric Library: http://gbdi.icmc.usp.br/arboretum/"

[2] "Google Dataset: http://www.google.com/programming-contest"

[3] Baeza Yates, R. A., and Navarro G.,"Fast approximate string matching in a dictionary.", *SPIRE Proceedings of the 5th International Symposium on String Processing and Information Retrieval,* 4:145, 1986.

[4] Bozkaya T, Ozsoyoglu M., "Indexing large metric spaces for similarity search queries.", *ACM Transactions on Database Systems,* 24(3):361-404, 1999.

[5] Brin S., "Near neighbor search in large metric spaces.", *VLDB Journal,* pp. 574-584, 1995.

[6] BrisaBoa R. N., Farina A., Pedreira O., "Sparse Selection of Sparse Pivots for Similarity Search in Metric Spaces", *SOFSEM 07,* pp. 8-13, 2007.

[7] Buckard W. A. and Keller R. M., "Some approaches to best-match file searching.", *ACM Commun.,* 16(4):230-236, 1973.

[8] Bustos B., Navarro G., Chavez E., "Pivot Selection Techniques for Promixity Searhing in Metric Spaces", *SCCC Proceedings of the 21. Conference of the Chilean Computer Science Society,* pp. 33-40, 2001.

[9] Celik C., "Priority Vantage Point Structures for Similarity Queries in Metric Spaces", *Proceedings of EurAsia-ICT,* vol. 2510 of LNCS, pp. 256-263, 2002.

[10] Celik C., "New Approaches to Similarity Searching in Metric Spaces". *PhD. Thesis.*

[11] Chavez E, Marroquin l. J. and Baeza-Yates R. A., "Spaghettis: An array based algorithm for similarity queries in metric spaces.", *In Proceedings of the 6th International Symposium on String Processing and Information Retrieval and International Workshop on Groupware (SPICE/CRIWG 1999),* pp. 38-46 , IEEE Computer Society, Cancun Mexico,1999.

[12] Chavez E, Marroquin l. J. and Navarro G., "Fixed Queries Array: A fast and economical data structure for proximity searching.", *Multimedia Tools Appl.,* 14(2):113-135 ,2001.

[13] Chavez E., Navarro G., Yates B. R., Marroquin L. "Searching in metric spaces.", *ACM Computing Surveys*, 33(3):273-321, 2001.

[14] Ciaccia P., Patella M. and Zezula P. "M-tree: An efficient access method for similarity search in metric spaces." *Proceedings of the 23rd International Conference on VLDB*, pp. 426-435, August. 1997.

[15] Ciaccia P., Patella M. "Bulk Loading M-Tree" *Proceedings of the 9th Australasian Database Conference (ADC)*, Australian Computer Science Communications, pp. 18-21, January. 1999, Springer.

[16] Ciaccia P. and Patella M., "The M^2-tree: Processing complex multi-feature queries with just one index." *In the Proceedings of the First DELOS Network of Excellence Workshop on Information Seeking,* December 2000.

[17] Comer D., "The Ubiquitous B-Tree", *ACM Computing Surveys*, 11(2):121-137 ACM Press.

[18] Filho S. F. R., Traina A., Traina Jr. C. , Faloutsos C., "Similarity Search without Tears: the OMNI-Family of All-Purpose Access Methods", *ICDE Proceedings of the 17th International Conference on Data Engineering,*pp. 620-630, 2001.

[19] Hafner J. L., Sawhney H. S., Equitz W., Flickener M., Niblack W."Efficient Color Histogram Indexing for Quadratic Form Distance Functions", *IEEE Transactions on Pattern Analysis and Machine Intelligence(TPAMI 1995)*, 17(7):729-736, IEEE Computer Society.

[20] Hjaltson R. G., Samet H., "Index-Driven Similarity Search In Metric Spaces.", *ACM Trans. Database Syst.*, 28(4):517-580, 2003.

[21] Huttenlocher D. P., Klanderman G. A., Rucklidge W. J. "Comparing Images Using Hausdorff Distance", *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI 1993)*, 15(9):850-863. IEEE Computer Society.

[22] Guttman A., "R-Trees: A dynamic index structure for spatial searching.", *Proceedings of the ACM International Conference on Management of Data(SIGMOD 1984)*, Boston, MA, USA, June 18-21, 1984, pp. 47-57, ACM Press.

[23] Kalantari I and McDonald G., "A data structure and an algorithm for the nearest point problem", *IEEE Transactions on Software Engineering,* 9(5):631-634, 1983.

[24] Kelly J. L. "General Topology.", 1955.

[25] Kruskal J. B. (1956), "On the shortest spaning subtree of a graph and the traveling salesman problem.", *In Proceedings of the American Mathematical Society*, volume 7, pp 48-50

[26] Lee D., "Query Relaxation for XML Model", *PhD Thesis*, University of LA, California USA.

[27] Levenshtein. V. I., "Binary Codes Capable of Correcting Spurious insertions and deletions of ones.", *Problems of Information Transmission*, 1:8-17, Kluwer Academic Publishers

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

[29] Navarro G., "Searching in metric spaces by spatil approximation.", *SPIRE Proceedings of the 6th International Symposium on String Processing and Information Retrieval,* pp: 141-148, 1999.

[30] Sankoff D., Kruskal J. B. "Time Warps, String Edits, and Macromolecules.", *Addison Wesley,* Reading Mass., 1983.

[31] Skopal T., "pivoting M-tree: A metric access method for efficient similarity search", *DATESO 2004 Proceedings of the annual International Workshop on Databases,* 2004.

[32] Traina Jr. C., Traina J. M. A., Seeger B., Faloutsos C. "Slim-trees: High performance metric trees minimizing overlap between nodes.", *EDBT 2000, 7th International Conference on Extending Database Technology*, pp. 27-31, March 2000.

[33] Traina Jr. C., Traina A., Filho S. F. R., Faloutsos C., "How to Improve the Pruning Ability of Dynamic Metric Access Methods", *CIKM,* pp. 219 - 226, 2002.

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

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

[36] Vidal E., "New Formulation and Improvements of the Nearest Neighbour Approximating and Eliminating Search Algorithm (AESA)",*Pattern Recognition Letters,*15(1):1-7, Elsevier, 1994.

[37] Yianilos P. N., "Data structures and algorithms for nearest neighbor search in general metric spaces.", *SODA Proceedings of the 4th Annual ACM Symposium on Discrete Algorithms,* pp: 311-321, 1993.

[38] Zezula P., Amato G., Dohnal V., Batko M., "Similarity Search: The Metric Space Approach", Springer, 2006.

[39] Zhou X., Wang G., Yu J. X., Yu G., "M^+ tree: A new dynamical multidimensional index for metric spaces.", *Proceedings of the 14th Australasian Database Conference on Database Technologies,* volume 17, pp: 161-168, 2003.