On Data Partitioning in Tree Structure Metric-Space Indexes

Rui Mao[1], Sheng Liu[1], Honglong Xu[1], Dian Zhang[1,*], and Daniel P. Miranker[2]

[1] Guangdong Province Key Laboratory of Popular High Performance Computers
College of Computer Science and Software Engineering, Shenzhen University
Shenzhen, Guangdong 518060, China
{mao,liusheng,2100230221,zhangd}@szu.edu.cn
[2] Department of Computer Sciences, University of Texas at Austin
Austin, TX 78712, USA
miranker@cs.utexas.edu

Abstract. Tree structure metric-space indexing methods recursively partition data according to their distances to a set of selected reference points (also called pivots). There are two basic forms of data partitioning: ball partition and General Hyper-plane (GH) partition. Most existing work only shows their superiority experimentally, and little theoretical proof is found. We propose an approach to unify existing data partitioning methods and analyze their performance theoretically. First, in theory, we unify the two basic forms of partitioning by proving that there are rotations of each other. Second, we show several theoretical or experimental results, which are able to indicate that ball partition outperforms GH partition. Our work takes a step forward in the theoretical study of metric-space indexing and is able to give a guideline of future index design.

Keywords: data partitioning, metric-space access method, pivot space model.

1 Introduction

Metric-space indexing, also known as distance-based indexing, is a general purpose approach to support similarity queries [9, 17, 29, 37]. It only requires that similarity is defined by a metric-distance function. Tree structure is one of the most popular metric-space indexing structures. In their top-down construction, tree structure metric-space indexing methods build index trees by recursively applying two basic steps: pivot selection and data partition. In pivot selection, only a small number of reference points (pivots), are selected from the database. The distances from data points to the pivots form a projection from the metric space to a low dimensional space, the pivot space [9, 24]. In data partitioning, data points are partitioned by their distances to the pivots, similar to the partitioning methods of multi-dimensional indexing [29].

Basically, there are two kinds of data partition strategies, ball partition and General Hyper-plane (GH) partition. However, most traditional methods are based on

* Corresponding author.

S.S. Bhowmick et al. (Eds.): DASFAA 2014, Part I, LNCS 8421, pp. 141–155, 2014.

heuristics. Theoretical analysis is usually overlooked. As a result, a large amount of these work only show their superiority from some carefully chosen datasets.

To solve this problem, we propose an approach that is able to unify ball and GH partitions, and compare their index performance theoretically and experimentally.

First, we show that ball partition and GH partition can be unified under the pivot space model [24] (Section 3.1), which was also proposed by our group. In the pivot space, the ball partition boundary is a straight line parallel to the axe, while GH partition boundary is a straight line with slope 1. That is, they are rotations of each other. Moreover, to make the analysis comprehensive, we leverage the extension versions of typical partition approach for comparison. In detail, we propose the Complete General Hyper-plane Tree (CGHT), which is an extension of the primitive form of GH partition, the GHT [33]. Results show that CGHT outperforms GHT.

Second, we propose an approach to analyze and predict an index's performance by means of the number of data points in a neighborhood (defined as *r*-neighborhood in Section 4) of the partition boundary. That is, less number of points in the *r*-neighborhood means better query performance. Further, we show two theoretical and two experimental results, which indicate that ball partition outperforms GH partition in query performance. They are: (1) among all the rotations, we theoretically prove that ball partition has the minimal width of *r*-neighborhood. Although the number of points in the *r*-neighborhood, or the size of the *r*-neighborhood, is jointly dominated by width and data density along the width, a minimal width is an indication of smaller size; (2) if data follow normal distribution in the 2-dimensional pivot space, we theoretically prove that ball partition has smaller *r*-neighborhood size than GH partition; (3) we experimentally show that MVPT has smaller *r*-neighborhood size than CGHT on a comprehensive test suite; (4) we experimentally show that MVPT has better query performance than CGHT on the comprehensive test suite.

The remaining of the paper is organized as follows. Related work is introduced in Section 2. In Section 3, we show how ball partition and GH partition are unified, followed by 2 theoretical indications of MVPT's better query performance than CGHT in Section 4. Section 5 elaborates the two experimental indications of MVPT's better query performance. Finally, conclusions and future work are discussed in Section 6.

2 Related Work

Ball partition and GH partition are the two basic kinds of data partition strategies in tree structure metric-space indexes.

Ball partition considers pivots one at a time, e.g. Vantage Point Tree (VPT) [33, 36] and Multi-Vantage Point Tree (MVPT) [5]. That is, data is first partitioned according to their distances to the first pivot. Each partition boundary forms a circle. Then, each partition area is further partitioned according to the data distances to the second pivot. Such process repeats until every pivot is used. For example, given k pivots and m partitions for each pivot, the total number of partitions is m^k.

GH partition is similar to clustering. Each data point is assigned to its closest pivot (or cluster center). In other words, the distance difference from a data point to two

pivots is considered, and the metric space is partitioned by a surface equidistant from the two pivots. General Hyper-plane Tree (GHT) [33], GNAT [6] and SA-tree [28] belong to GH partition.

The BSP [16], M-Tree [9] and slim-tree [10] family also adopt the GH partition with multiple pivots when data is partitioned. However, each partition is represented as a ball, i.e. a center and a covering radius, similar to ball partition.

3 Unification of Ball Partition and GH Partition

In this section, we first give a brief introduction to our theoretical background, the pivot space model. Then, we consider VPT and GHT, the primitive forms of ball partition and GH partition. However, VPT has only one pivot while GHT has two pivots. To make the numbers of pivots and partitions equal as a fair comparison, we propose the Complete General Hyper-plane Tree (CGHT). Finally, we compare CGHT with MVPT, with same numbers of pivots and partitions, and unify them in the pivot space.

3.1 Theoretical Background: The Pivot Space Model

This subsection presents a brief introduction to the pivot space model. The readers can refer to [24] for more details.

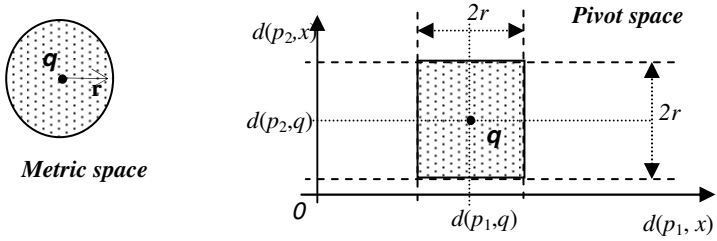

Fig. 1. In a metric space, the ball of a range query is covered by a square in the pivot space

Some notations that will be used in the paper are listed as follows. Let R^m denotes a general real coordinate space of dimension m. Let (M, d) be a metric space, where M is the domain of the space, and d is a metric distance function. Let $S = \{x_i \mid x_i \in M, i = 1, 2, \ldots, n\}$, be the database, and x_i is the data point in the database, $n \geq 1$. S is a finite indexed subset of M. Let $P = \{p_j \mid j = 1, 2, \ldots, k\}$ be a set of k pivots. $P \subseteq S$. Duplicates are not allowed.

There are three steps to answer a range query $R(q, r)$ in a metric space [24]:

Step 1: (1) Map the data into R^k. (2) Map the query object into R^k. (3) Determine a region in R^k that completely covers the range query ball.

Given the set of pivots, each point x in S can be mapped to a point x_p in the non-negative orthant of R^k. The j-th coordinate of x_p represents the distance from x to p_j:

$$x_p = (d(x, p_1), \ldots, d(x, p_k)),$$

The pivot space of S, $F_{P,d}(S)$, is defined as the image set of S:

$$F_{P,d}(S) = \{x_p \mid x_p = (d(x,p_1), \ldots, d(x,p_k)), x \in S\}.$$

The image shape of a range query ball $R(q, r)$ in a general metric space is not clear in a corresponding pivot space. However, it can be proved, from the triangle inequality, the image of the query ball is completely covered by a hypercube of edge length $2r$ in the pivot space [9]. Actually, the hypercube is a ball of radius r in the new metric space specified by the pivot space and the L^∞ distance. We call the hypercube the query cube. Fig. 1 shows an example where 2 pivots are selected [24]. All the points in the query ball are mapped into the query square in the 2-d pivot space plane. Points outside the square can be safely discarded according to the triangle inequality.

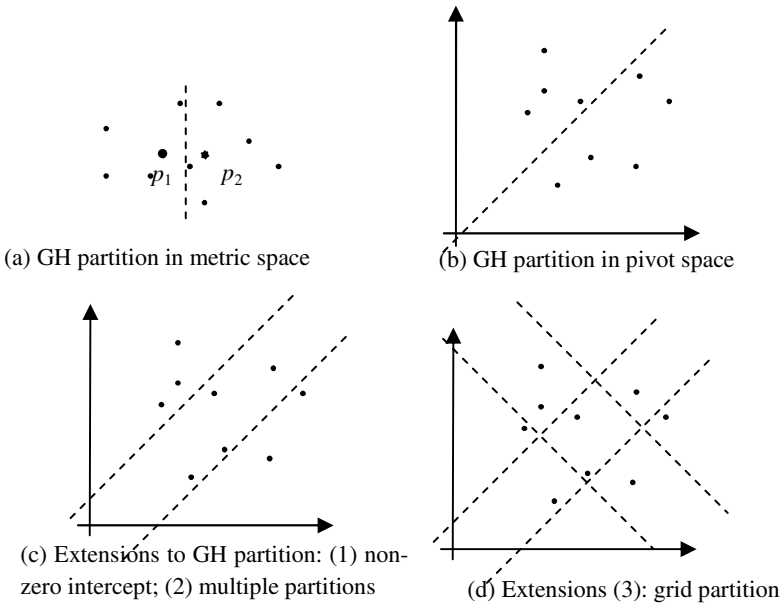

Fig. 2. GH partition and extensions

Step 2: Exploit multi-dimensional techniques to obtain all the points in the region determined in Step 1.

Fundamentally, Step 2 uses divide-and-conquer, based on the data coordinates. Many partition methods in multi-dimensional indexing can be applied here. For example, MVPT's partition is similar to a kd-tree [2].

Step 3: For each point obtained in Step 2, we compute its distance to the query object to eliminate the false positives.

3.2 The Complete General Hyper-plane Tree

We start with the primitive form of ball partition and GH partition for comparison. VPT [33, 36] is the primitive form of ball partition. It first selects a pivot, and then draws a circle centered at the pivot to partition the data. The radius of the circle is chosen so that the partitioning is balanced. GHT [33] is the primitive form of GH partition. It first selects two pivots. Then it uses a hyper-plane, which is equidistant to the two pivots, to partition the data. Obviously, VPT and GHT use different number of pivots. Since GH partition uses at least two pivots, we consider Multiple Vantage Point Tree (MVPT) [5] instead of VPT as a representative of ball partition methods. MVPT first selects a few pivots. Then data is partitioned based on their distances to the first pivot. Such partitioning repeats until all the pivots are used.

A MVPT with 2 pivots has the same number of pivots with a GHT. However, at each index node, a MVPT with 2 pivots generates at least 4 partitions (2 by 2), while a GHT generates only 2. To make the number of partitions equal, we extends GHT and propose the CGHT, which has the same numbers of pivots and partitions with MVPT with 2 pivots.

In the following, for simplicity, we assume only two pivots are selected, $k = 2$. Let d_1 and d_2 be the distances from a data point to the two pivots, respectively.

Given the two pivots, a GH partition separates the data with a hyper-plane formed by points equidistant to the two pivots (Fig. 2 (a)) [33]. In the pivot space, the partition line is specified by $d_1 - d_2 = 0$. Its slope is 1 and its intercept is 0 (Fig. 2(b)). Examining Fig. 2(b), we can extend GHT as follows.

Extension (1): use non-zero intercept (Fig. 2(c)).
Extension (2): use multiple partitions (Fig. 2(c)).
Extension (1) has been studied by Zhang and Li, and Lokoc and Skopal [23, 38].

In GHT, distances of each data point to two pivots are computed. So the two variables, d_1 and d_2, are known. However, only one combined variable, $d_1 - d_2$, is exploited in partitioning. In statistics, if d_1 and d_2 are two independent random variables, $d_1 - d_2$ and $d_1 + d_2$ are deemed independent [8]. Although d_1 and d_2 might not be actually independent, we believe that variable $d_1 + d_2$ is able to provide additional pruning ability without extra distance calculations. This idea leads to Extension (3):

Extension (3): use $d_1 + d_2$ to partition.

***p*₁**: pivot 1
***p*₂**: pivot 2
plus_max[]: array of all children's upper bounds of $d_1 + d_2$
plus_min[]: array of all children's lower bounds of $d_1 + d_2$
minus_max[]: array of all children's upper bounds of $d_1 - d_2$
minus_min[]: array of all children's lower bounds of $d_1 - d_2$
children[]: array of pointers to all children

Fig. 3. Internal node structure of CGHT

Fig. 2 (d) shows the case that all the 3 extensions are combined. As Extension (3) exploited all the information provided by the two pivots, we name this new structure the Complete General Hyper-plane Tree (CGHT). Here "Complete" means the two distance computation to the two pivots are taken into full use. That is, during the search procedure, in addition to pruning based on one variable d_1 - d_2, one can also do pruning base on another variable, d_1 + d_2. The structure of one CGHT internal node is shown in Fig. 3. The structure of one CGHT leaf node is the same as that of GHT [33].

(1) if data set is small, construct a leaf node; otherwise:
(2) pivot selection
(3) compute $d_1 + d_2$ and $d_1 - d_2$ for all data points
(4) run clustering partition on $d_1 + d_2$ and $d_1 - d_2$ values.
(5) recursively bulkload every partitions generated in (4).

Fig. 4. Steps to bulkload a CGHT

Bulkloading of CGHT follows the general steps in constructing a pivot-based index structure (Fig. 4). First, two pivots are selected, using any pivot selection algorithm. Second, values of $d_1 + d_2$ and $d_1 - d_2$ are computed for each data point. Third, most data partition algorithms (from the three categories of metric space indexing methods) can be applied to partition the data, based on $d_1 + d_2$ and $d_1 - d_2$ values. These steps are run recursively on each partition generated.

(1) if leaf node, do the GHT leaf node search; otherwise
(2) compute $d_1 + d_2$ and $d_1 - d_2$ values for the query object
(3) for each child i,
if not [(plus_min[i]$-2r \le d_1+d_2 \le$ plus_max[i]$+2r$) and
(minus_min[i]$-2r \le d_1-d_2 \le$ minus_max[i]$+2r$)]
then this child can be pruned.
(4) recursively search child that cannot be pruned in (3).

Fig. 5. Range query steps of CGHT

Range query algorithm of CGHT is summarized in Fig. 5. The correctness of the pruning step (3), can be proven by the triangle inequality.

3.3 Unification of CGHT and MVPT

The CGHT partition lines with slope 1 in the pivot space (Fig. 2(d)) are actually hyperbolas in metric space (Fig. 6(a)), while those with slope -1 are actually ellipses. For the MVPT partition circles in the metric space (Fig. 6(b)), in the pivot space (Fig. 6(c)), they are actually straight lines parallel to the axes.

Comparing Fig. 2(d) with Fig. 6(c), it is obvious that one can turn MVPT into CGHT by a 45^o rotation, or vice versa.

Now ball partition and GH partition are unified. A following reasonable question is what rotation angle is optimal with respect to query performance. We show 2

theoretical indications of ball partition's better query performance in Section 4, and 2 experimental indications in Section 5.

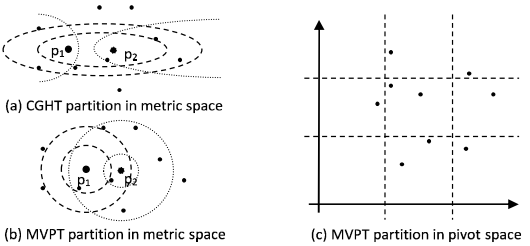

Fig. 6. CGHT and MVPT partitions

4 Theoretical Comparison of Ball Partition and GH Partition

In this section, we analyze ball partition and GH partition's query performance theoretically. We first define the *r*-neighborhood of the partition boundary, the number of points in which is a dominant factor of the query performance. Then, the total number of points in the neighborhood (size of *r*-neighborhood) is investigated for both ball partition and GH Partition from two aspects: (1) width of *r*-neighborhood, a dominative factor of *r*-neighborhood size; (2) *r*-neighborhood size for a particular data distribution, 2-dimensional normal distribution, in the pivot space. Our results indicate that MVPT outperforms CGHT.

4.1 *r*-Neighborhood of Partition Boundaries

Since the key of tree structure indexing is to prune as much data as possible, the pruning power is one of the dominant factors of query performance. Given a range query $R(q, r)$, if q is so close to a partition boundary that the query ball centered at q with radius r intersect the boundary, either sides of the boundary can be pruned, leading to low pruning power, or low query performance. To increase the pruning power, one has to lower the probability that queries fall too close to partition boundaries. As a result, we define the "*r*-neighborhood". In plain terms, an *r*-neighborhood is the neighborhood of a partition boundary, such that a range query $R(q, r)$ may intersect the boundary if q falls into the *r*-neighborhood.

Definition 1. The ***r*-neighborhood** of a partition boundary L, denoted as $N_r(L)$, is a neighborhood of L. If a query object q falls into it, the range query ball centered at q with radius r intersects L.

If q falls into $N_r(L)$, partitions comprising both halves of the space defined by L have to be searched. If q does not fall into $N_r(L)$, only the partition on the side of L ,where q falls into, need to be searched. The partition on the other side can be pruned.

Average query performance can be improved by reducing the probability of queries falling into *r*-neighborhood. Assume queries have the same distribution as the data

points in the database, the total number of data points in the *r*-neighborhood, or *r*-neighborhood size, is critical to query performance. That is, the smaller the *r*-neighborhood size, the better the query performance.

R-neighborhood is also useful when redundancy is introduced in the index tree. To guarantee that only one partition will be further considered at each level of the index tree, one can duplicate the *r*-neighborhood of the partition boundaries [14].

4.2 Minimal Width of the *r*-Neighborhood

The *r*-neighborhood size, a dominant factor of query performance, is jointly determined by the width of the *r*-neighborhood and the density of data along the width. Since density is determined by data distribution, it is impossible to know it when design the index. Therefore, in this subsection, we find out the minimal width of *r*-neighborhood among all rotations of ball partition and GH partition.

As discussed in Section 3, ball partition and GH partition boundaries are straight lines in *2-d* pivot spaces. Let line *L*: $y = ax$ be an arbitrary partition line rotated from ball partition or GH partition (without losing generality, the constant intercept is omitted). We name such type of partitioning **linear partition** in the pivot space. The shape of *r*-neighborhood of linear partition in the pivot space is given by Theorem 1.

Theorem 1. The *r*-neighborhood of line *L*: $y = ax$, $0 \le a$ is $N_r(L)$: $|y-ax| \le R_L(r)$, where $R_L(r)$ is a real valued function of *r* and *L*.

In other words, Theorem 1 states the fact that the *r*-neighborhood of linear partition is delimited by two lines, which are parallel and with the same distance to the linear partition line. The proof is straightforward and is omitted.

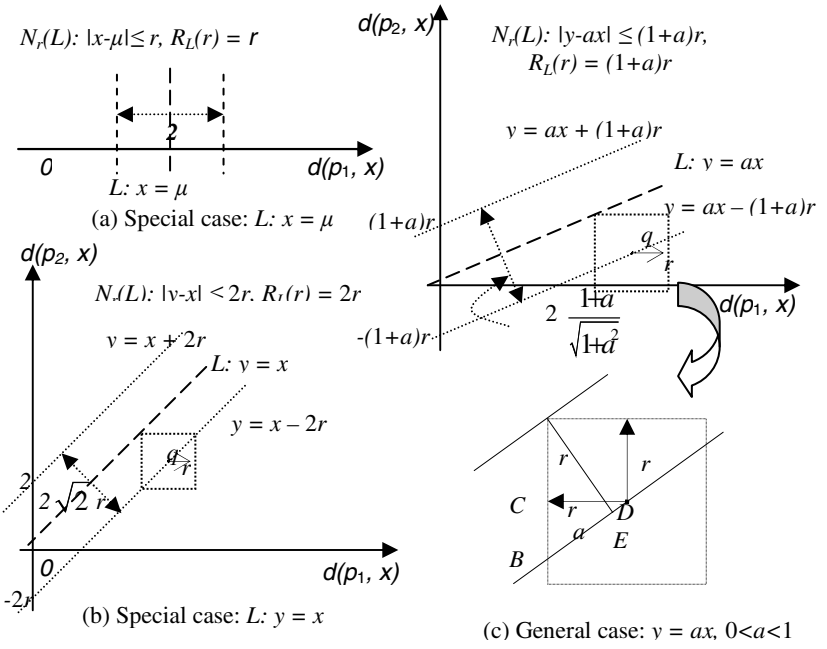

Fig. 7. R-neighborhood of hyper-plane partitions

In the following, we discuss the value of $R_L(r)$, and the width of $N_r(L)$. We start from two special cases, and followed by the general cases.

Case 1: L: $x = \mu$

This is the ball partition. From simple analysis (Fig. 7 (a)), the r-neighborhood of L: $x = \mu$ is: $N_r(L) : |x-\mu| \le r$, $R_L(r) = r$, and the width of $N_r(L)$ is $2r$.

Case 2: L: $y = x$

This is the GH partition. According to the pivot space model, the query ball in metric space is covered by a cube in the pivot space. It can be observed (Fig. 7 (b)) that the r-neighborhood of L: $y = x$ is:

$N_r(L)$: $|y-x| \le 2r$, $R_L(r) = 2r$, and the width of $N_r(L)$ is $2\sqrt{2}\,r$.

Case 3: General case: $y = ax$

Because of the symmetry, we only need to consider the case that $0<a<1$. The result is stated in Theorem 2.

Theorem 2. Let L: $y = ax$, $0<a<1$, then $R_L(r) = (1+a)r$, and the width of $N_r(L)$ is $2\frac{1+a}{\sqrt{1+a^2}}\,r$.

Theorem 3 is illustrated in Fig. 7(c). It can be proved with basic geometry.

The following corollary is correct due to the fact that $0 \le a$.

Corollary 1. The width of $N_r(L)$ is minimized to $2r$ when $a = 0$ *or* ∞, i.e. the partition line is parallel to the axes, or ball partition. The width of $N_r(L)$ is maximized to $2\sqrt{2}\,r$, when $a=1$, i.e. the partition line has 45° angle to the axes, or GH partition.

According to Corollary 1, ball partition is optimal among its rotations with respect to the width of r-neighborhood. Although the r-neighborhood size is determined by both the width and the density, a smaller width is still an important sign of fewer points in the r-neighborhood, thus better query performance. To further investigate the r-neighborhood size, next, we mathematically calculate it for a particular data distribution, 2-dimensional normal distribution, in the pivot space.

4.3 *r*-Neighborhood Size of *2-d* Normal Distribution in Pivot Space

Let us consider the case where data is normally distributed in a 2-d pivot space (Fig. 8). For simplicity, let the joint distribution of d_1 and d_2 to be $N(0, 1, 0, 1, -\rho)$. That is, the marginal distribution of both d_1 and d_2 is the standard normal distribution, and correlation coefficient is $-\rho$, $0 \le \rho \le 1$.

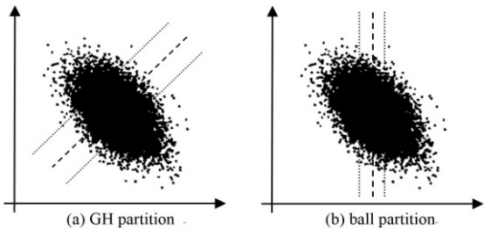

Fig. 8. Example: normal distribution in *2-d* pivot space

In the following, we calculate the numbers of points in the *r*-neighborhoods of ball and GH partitions, denoted by $|N_{Lb}(r)|$ and $|N_{LG}(r)|$, respectively. Let $P_{Lb}(r)$ and $P_{LG}(r)$ be the probability that the data falls into the *r*-neighborhoods of ball and GH partitions. It is sufficient to compare $P_{Lb}(r)$ with $P_{LG}(r)$.

Theorem 3. $P_{Lb}(r) \leq P_{LG}(r)$, and they equal when $\rho = 1$.

Proof: In statistics, it have been proved that for normal distribution $N(0, 1, 0, 1, -\rho)$, the projected distribution on the first principle component (perpendicular to the GH partition line in Fig.8(a)) is $N(0, 1+\rho)$, normal distribution with variance $1+\rho$ [20].

From Theorem 2, the widths of *r*-neighborhoods of ball and GH partitions are $2r$ and $2\sqrt{2}\,r$, respectively. Therefore:

$P_{Lb}(r) = P(|x| \leq r \mid x \sim N(0,1))$, and $P_{LG}(r) = P(|x| \leq \sqrt{2}\,r, \mid x \sim N(0, 1+\rho))$

After normalization: $P_{LG}(r) = P(|x| \leq \sqrt{2/(1+\rho)}\,r, \mid x \sim N(0, 1))$

Since $0 \leq \rho \leq 1$, we have $1 \leq \sqrt{2/(1+\rho)}$, and the equality holds when $\rho = 1$.

That is, $P_{Lb}(r) \leq P_{LG}(r)$, and the equality holds when $\rho = 1$. □

In conclusion, for *2-d* normal distribution in the pivot space, better query performance can be expected from ball partition than GH partition since there are fewer points in the *r*-neighborhood of ball partition boundary than those of GH partition.

5 Empirical Results

In this section, we present experimental results to indicate that ball partition outperforms GH partition. We first introduce a comprehensive test suite we use, then the *r*-neighborhood sizes of both partitions on the test suite, and finally the range query performance of both partition methods on the test suite.

Table 1. Summary of test suite

Workload	Db. size	Distance metric	Dom.dim.	radius
Uniform vector	1Million		2, 8	0.02/0.3
Texas	36463	L^2 norm	2	0.02
Hawaii	9290		2	0.02
Protein	100k	Weighted edit distance	18	2
DNA	100k	Hamming distance	6	2
Image	10,221	*L*-norms	66	0.02
English dictionary	69069	Edit distance	N/A	1
NASA image	40150	L^2 norm	20	0.03

5.1 Test Setup

Our test data consists of the MoBIoS test suite [27] and the SISAP test suite [31] (Table 1). The MoBIoS test suite consists of synthetic vector data (dimensions 2 and

8), biological data, real world vector data and an image dataset. Different dimensions of the synthetic vector data have independent identical uniform distributions. Two types of biological data are considered: (1) amino-acid sequence fragments of the yeast proteome with weighted-edit distance based on the metric PAM substitution matrix [36]; (2) the DNA sequence fragments of the Arabidopsis genomes with Hamming distance. The real world vector data consists of the US cartographic boundary data of Texas and Hawaii. The images are represented by 66 dimensional feature vectors with a linear combination of L^1 and L^2 norms. Two datasets from the SISAP test suite [31] are involved: the English dictionary uses the edit distance. The NASA images are represented by 20-dimensional vectors with the Euclidean distance.

The sizes of the databases are all 100*k*, except for uniform vector, which is 1 million, and those small workloads, where only limited amount of data is available. Although the data volume is not huge, it is large enough to clearly show the trends.

The number of pivots is 2 for all cases. Adding more pivots will increase the dimension of the pivot space. In Section 3, we have shown that ball partition and GH partition are rotations of each other. The only difference between them is the angle of rotation. Follow this idea, in the multiple-pivot scenario, one can still rotate the partition boundary of ball partition to get GH partition, or vice versa. This transformation is similar to the linear transformation of bases of vector space in linear algebra. Therefore, we limit our number of pivots to 2 for simplicity.

Two pivot selection heuristics are examined. One is *Farthest-First-Traversal* (FFT) [18], which selects corner of data as pivots. The other is a PCA-based heuristic, which has been shown to perform well generally [24].

For each pivot, data is partitioned into 3 parts. Thus, the total number of partitions is 9. Two data partition heuristics are examined. One is balanced partition [9], and the other is *Clusteringkmeans*, which derives the partition from *k*-means clustering [25]. Please note that GHT only adopts the balanced partition.

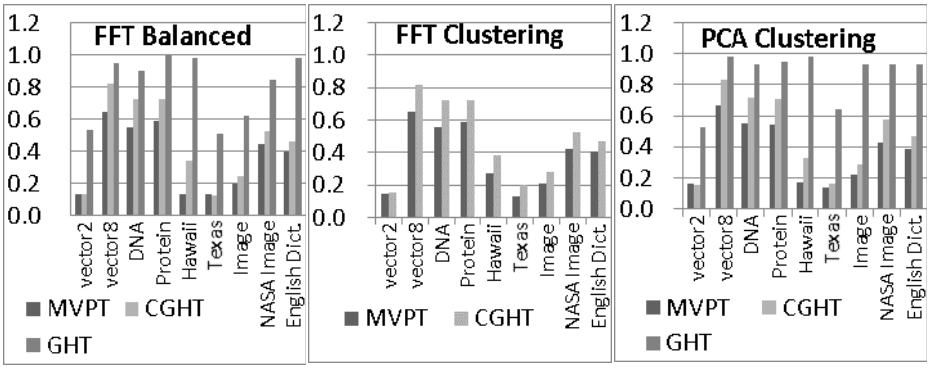

Fig. 9. Percentage of data to further examine at the index root

For each dataset, combination of pivot selection and data partition heuristics, statistics with a number of range query radii are collected. However, to save space, only the results of one representative radius are presented for each dataset and combination of heuristics. Those representative radii are listed in Table 1. For each dataset, 5000 data objects are chosen sequentially from the beginning of the dataset files as range query objects. The experimental results collected are averaged over queries.

We first compare the *r*-neighborhood size of MVPT, GHT and CGHT, then the range query performance.

5.2 Number of Points in *r*-Neighborhood

This subsection examines the index performance only at the tree root level, and the whole index is examined in the next subsection, based on range query performance.

In the presence of multiple pivots, a query in the *r*-neighborhood might lead to more than 2 partitions being further search. Therefore, instead of simply counting the number of points in *r*-neighborhood, we collect the average percentage of data to further examine, or the data that cannot be pruned, at the index root, as a more accurate indicator of the query performance.

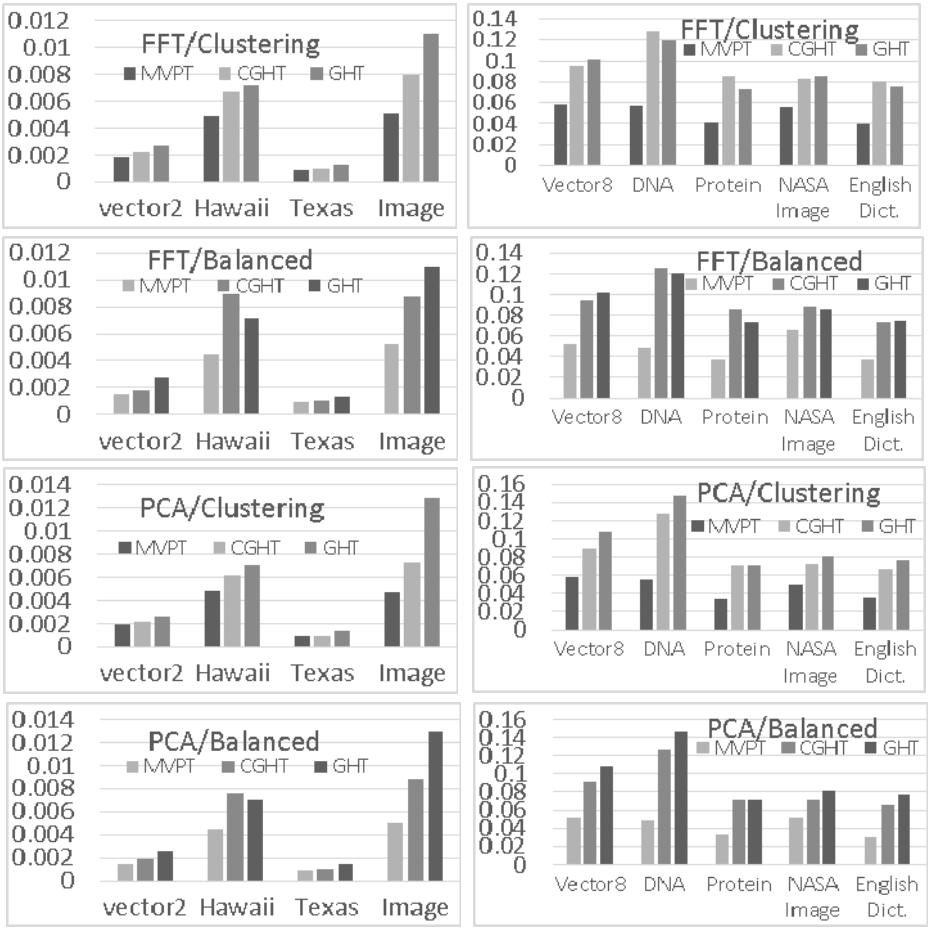

Fig. 10. Query performance of MVPT, CGHT and GHT

Fig. 9 shows the percentage of data cannot be pruned, or pruning power, at the index root level. Obviously, CGHT has more pruning power than GHT in all the cases, indicating a significant improvement of CGHT over GHT. Furthermore, MVPT has more pruning power than CGHT for almost all the cases. The only two exceptions are for Texas data (FFT, Balance) and 2-dimensional vector (PCA, Clustering), where the difference is neglectable.

5.3 Range Query Performance

Since distance evaluation in a metric space is usually costly, to focus on the algorithmic behavior, we use the number of distance calculations normalized over dataset sizes, which is implementation independent, as a performance measure.

The range query performance of MVPT, CGHT and GHT are listed in Fig. 10. It is clear to see that MVPT outperforms CGHT and GHT. The differences are significant in most cases. Moreover, the performance is more than doubled in several cases.

In the total 36 comparisons of CGHT and GHT, CGHT outperforms GHT in 28 cases. However, there are 8 of 36 cases that GHT outperforms CGHT, while the differences are significant in a few cases. We would like to point out that it does not prevent one from concluding that MVPT generally outperforms CGHT or GHT, or, in other words, ball partition generally outperforms GH partition.

6 Conclusions and Future Work

In this paper, we unify data partition methods of tree structure metric-space indexes under the pivot space model, and propose an approach to compare and predict query performance through *r*-neighborhood size. Theoretical and experimental results indicate that ball partition outperforms GH partition.

The contributions of our paper are as follows:

(1) Propose the Complete General Hyper-plane Tree which takes full use of the pivots and outperforms GHT.

(2) Unify ball partition and GH partition in the context of the pivot space model.

(3) Show that ball partition possesses the optimal rotation angle with respect to the width of *r*-neighborhood.

(4) Prove that ball partition has fewer points in *r*-neighborhood than GH partition for *2-d* normal distribution in the pivot space.

(5) Show that ball partition has fewer points in *r*-neighborhood than GH partition, based on our test benchmark.

(6) Show that MVPT outperforms CGHT by experiments, which consists with the theoretical analysis.

The methodology developed in this paper forms a basis for resolving the merits of the different classes of metric-space indexing algorithms.

The pivot space model establishes a bridge connecting metric-space indexing and high-dimensional indexing. It also lays a foundation for theoretical analysis. In the future work, we plan to take advantage of the fruitful results along the much longer

research history in high-dimensional indexing to study metric-space indexing. We believe some of the dimension reduction and data partition approaches in high-dimensional indexing may benefit to metric-space indexing. Moreover, we have only considered linear partition so far. Non-linear partitioning deserves more attention. Actually, there are already some efforts on this topic, such as Li and Zhang's Haperplane tree [22].

Acknowledgment. This research was supported by the following grants: China 863: 2012AA010239; NSF-China: 61033009, 61170076, U1301252, 61202377, 61103001; Shenzhen Foundational Research Project: JCYJ20120613161449764.

References

1. Aizerman, M., Braverman, E., Rozonoer, L.: Theoretical foundations of the potential function method in pattern recognition learning. Automation and Remote Control 25, 821–837 (1964)
2. Bentley, J.L.: Multidimensional binary search trees used for associative searching. ACM Commun. 18(9), 509–517 (1975)
3. Beygelzimer, A., Kakade, S., Langford, J.: Cover trees for nearest neighbor. In: ICML, pp. 97–104 (2006)
4. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Haussler, D. (ed.) 5th Annual ACM Workshop on COLT, pp. 144–152. ACM Press, Pittsburgh (1992)
5. Bozkaya, T., Ozsoyoglu, M.: Indexing large metric spaces for similarity search queries. ACM Trans. Database Syst. 24(3), 361–404 (1999)
6. Brin, S.: Near Neighbor Search in Large Metric Spaces. In: The 21th International Conference on Very Large Data Bases (VLDB 1995). Morgan Kaufmann Publishers Inc. (1995)
7. Bustos, B., Navarro, G., Chavez, E.: Pivot selection techniquesfor proximity searching in metric spaces. Pattern Recogn. Lett. 24(14), 2357–2366 (2003)
8. Casella, G., Berger, R.L.: Statistical Inference. Duxbury Press (2001)
9. Chavez, E., Navarro, G., Baeza-Yates, R., Marroqu, J.: Searching in metric spaces. ACM Computing Surveys 33(3), 273–321 (2001)
10. Ciaccia, P., Patella, M.: Bulk loading the M-tree. In: 9th Australasian Database Conference, ADO 1998 (1998)
11. Ciaccia, P., Patella, M.: Proceedings of the Third International Conference on Similarity Search and Applications, Istanbul, Turkey, September 18-19. ACM, New York (2010)
12. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. Presented at the 23rd International Conference on Very Large Data Bases (VLDB 1997), Athens, Greece (1997)
13. Cortes, C., Vapnik, V.: Support-Vector Networks. Machine Learning 20 (1995)
14. Feng, Y.-C., Kui, C., Cao, Z.-S.: A Multidimensional Index Structure for Fast Similarity Retrieval. Journal of Software 13(8), 1678–1685 (2002)
15. Filho, R.F.S., Traina, A.J.M., Traina, C., Faloutsos, C.: Similaritysearch without tears: The OMNI family of all-purpose accessmethods. In: ICDE, pp. 623–630 (2001)
16. Fuchs, H., Kedem, Z.M., Naylor, B.F.: On visible surface generation by a priori tree structures. In: Proceedings of the 7th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1980), SIGGRAPH Computer Graphics, vol. 14(3), pp. 124–133. ACM, New York (1980)

17. Hjaltason, G.R., Samet, H.: Index-driven similarity search in metric spaces. ACM Transactions on Database Systems (TODS) 28(4), 517–580 (2003)
18. Hochbaum, D.S., Shmoys, D.B.: A best possible heuristic for the k-center problem. Mathematics of Operational Research 10(2), 180–184 (1985)
19. Jagadish, H.V., Ooi, B.C., Tan, K., Yu, C., Zhang, R.: iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. ACM Trans. Database Syst. 30(2), 364–397 (2005)
20. Johnson, R.A., Wichern, D.W.: Applied Multivariate Statistical Analysis, 6th edn. Prentice Hall (2007)
21. Karger, D., Ruhl, M.: Finding nearest neighbors in growth restricted metrics. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing, pp. 741–750 (2002)
22. Li, J.-Z., Zhang, Z.-G.: Haperplane Tree: A Structure of Indexing Metric Spaces for Similarity Search Queries. Journal of Computer Research and Development 40(8), 1209–1215 (2003)
23. Lokoč, J., Skopal, T.: On applications of parameterized hyper-plane partitioning. Poster in the Proceedings of the Third International Conference on Similarity Search and Applications (SISAP2010), Istanbul, Turkey, September 18-19, pp. 131–132 (2010)
24. Mao, R., Miranker, W., Miranker, D.P.: Pivot Selection:Dimension Reduction for Distance-Based Indexing. Journal of Discrete Algorithms 13, 32–46 (2012)
25. Mao, R., Xu, W., Ramakrishnan, S., Nuckolls, G., Miranker, D.P.: On Optimizing Distance-Based Similarity Search for Biological Databases. In: The 2005 IEEE Computational Systems Bioinformatics Conference (2005)
26. Matousek, J.: Lectures on Discrete Geometry, p. 497. Springer-Verlag New York, Inc. (2002)
27. MoBIoS test suite, `http://www.cs.utexas.edu/~mobios/`
28. Navarro, G.: Searching in Metric Spaces by Spatial Approximation. In: Proceedings of the String Processing and Information Retrieval Symposium & International Workshop on Groupware. IEEE Computer Society (1999)
29. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Morgan-Kaufmann (2006)
30. Shen, H.T., Ooi, B.C., Zhou, X.: Towards Effective Indexing for Very Large Video Sequence Database. In: SIGMOD Conference 2005, pp. 730–741 (2005)
31. SISAP test suite, `http://sisap.org/Metric_Space_Library.html`
32. Traina Jr., C., Traina, A.J.M., Seeger, B., Faloutsos, C.: Slim-trees: High performance metric trees minimizing overlap between nodes. In: Zaniolo, C., Grust, T., Scholl, M.H., Lockemann, P.C. (eds.) EDBT 2000. LNCS, vol. 1777, pp. 51–65. Springer, Heidelberg (2000)
33. Uhlmann, J.K.: Satisfying General Proximity/Similarity Queries with Metric Trees. Information Processing Letter 40(4), 175–179 (1991)
34. Venkateswaran, J., Kahveci, T., Jermaine, C.M., Lachwani, D.: Reference-based indexing for metric spaces with costly distance measures. VLDB J. 17(5), 1231–1251 (2008)
35. Xu, W., Miranker, D.P.: A Metric Model of Amino Acid Substitution. Bioinformatics 20(8), 1214–1221 (2004)
36. Yianilos, P.N.: Data structures and algorithms for nearest neighbor search in general metric spaces. In: The Fourth Annual ACM-SIAM Symposium on Discrete Algorithms.Society for Industrial and Applied Mathematics (1993)
37. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search: the Metric Space Approach. Springer, Heidelberg (2006)
38. Zhang, Z.-G., Li, J.-Z.: An Algorithm Based on RGH-Tree for Similarity Search Queries. Journal of software 13(10), 1969–1976 (2002)