

A New Indexing Method for High Dimensional Dataset

Jiyuan An¹, Yi-Ping Phoebe Chen¹, Qinying Xu² and Xiaofang Zhou³

¹ Deakin University, Australia

² University of Tsukuba, Japan

³ University of Queensland, Australia

{jiyuan, phoebe}@deakin.edu.au

qinying@kslab.is.tsukuba.ac.jp

zxf@itee.uq.edu.au

Abstract. Indexing high dimensional datasets has attracted extensive attention from many researchers in the last decade. Since R-tree type of index structures are known as suffering “curse of dimensionality” problems, Pyramid-tree type of index structures, which are based on the B-tree, have been proposed to break the curse of dimensionality. However, for high dimensional data, the number of pyramids is often insufficient to discriminate data points when the number of dimensions is high. Its effectiveness degrades dramatically with the increase of dimensionality. In this paper, we focus on one particular issue of “curse of dimensionality”; that is, the surface of a hypercube in a high dimensional space approaches 100% of the total hypercube volume when the number of dimensions approaches infinite. We propose a new indexing method based on the surface of dimensionality. We prove that the Pyramid tree technology is a special case of our method. The results of our experiments demonstrate clear priority of our novel method.

1 Introduction

Multimedia objects, such as images, video and audio clips, are often mapped into a high dimensional space, such that similarity search among these multimedia objects are translated into distance-based queries in the high dimensional feature space. To facilitate efficient search in very large amount of multimedia datasets, it is necessary to use a high dimensional index mechanism that must be able to scale not with the amount of the data, but more importantly, also with the number of dimensionality. R-tree and its variations are commonly used index methods for multi-dimensional datasets [8][6]. The basic idea of R-tree and its variations can be briefly described as the following. Firstly, a multi-dimensional space is partitioned recursively into a hierarchical structure according to data distribution. Secondly, the partitioned subspaces are permitted to overlap with each other. However, the idea of permitting overlapping sibling subspaces brings a serious drawback for high dimensional spaces, as the overlapping extent of sibling R-tree nodes in the directory increases rapidly to about 90% of their entire volume when the dimensionality is increased to 5 [4][1][2]. That defeats the purpose of hierarchical indexing completely, as most nodes can not be pruned in the searching process. This type of indexes also suffers another problem. The fan out of a node becomes very small due to the large size of coordi-

nates for high dimensional data. Consequently, the performance of such an index structure degrades and its effectiveness is sometimes worse than linear scan. Many improvements have been proposed (e.g.[4]), but the problems mentioned above still exist for most known attempts.

If an indexing method requires keeping all coordinates of data items in the index, the size of the index structure is, of course, proportional to the dimensionality. To reduce the size of index, Bechtold *et al* proposed a method called the Pyramid-tree [5] [11][13], where high dimensional data is mapped into a linear space such that some classical index structures, such as B⁺-tree, can be used. This results a better performance than the X-trees and other R-tree inspired multidimensional indexes. However, for one pyramid, it is known that most data items concentrate on its base. The slices in each pyramid can not make the index more discriminative, as the data points in one pyramid always have the same index value.

In a Pyramid tree, one data point is associated with one pyramid, and a data point is indexed by a base of pyramid. If the dimensionality is d , the base of pyramid is $(d - 1)$ -D hyperplane. In this paper, we generalize the ideal of using $(d - 1)$ -D hyperplane to using $(d - i)$ -D hyperplane ($1 \leq i \leq d$). The number of index values will be increased in such a way to make the index more discriminative for a better search performance. This paper, to the best of our knowledge, proposes the first solution to index high dimensional data based on surfaces. We will demonstrate its significantly improved efficiency comparing to the traditional Pyramid tree indexing. We also show that the Pyramid tree is a special case of our surface-based indexing method.

Section 2 of this paper describes the motivation of surface index structure. In Section 3, we discuss the structure of a novel surface-based spatial index method. In Section 4, we propose a method for the allocation of pyramids for the data points on the boundary. Section 5 shows the results of experiments comparing with the Pyramid-tree technique.

2 Motivation

Because of the limitation of visual, no one can see more than 3 dimensional spaces. We can only image them from 2- or 3-D space. However, we give some examples to show that high dimensional space is not imaginable from 2- or 3-D space. Throughout this paper, the data space is normalized into $(0, 1]$. So our objective data space is a hypercube. Its length of edges is 1.

2.1 Non-intuition of High Dimensionality

High dimensional data can be found everywhere. Time series is a traditional high dimensional data. We can also use high dimensional data to describe an object, such as, a people can be described using his features (tall, weight, age, and etc.). If two objects are similar, we consider they are corresponded to two near data points in high dimensional space even we can no be seen. The axes of the dimensional space consist of the feature vectors. However, high dimensional data has own characters which are different with data in 2- or 3-D

space. Figure 1(A) shows a circle of r radius and its circumscribed quadrilateral. The distance from a vertex of quadrilateral to circle is denoted by a , where a holds the following inequality in 2- or 3-D space.

$$r \geq a$$

However, as the dimension increases, this inequality becomes inappropriate. For example, in the case of 16-D space, a has 3 times length of r ($a=3r$). It can be calculated by the following equation.

$$(r + a)^2 = \underbrace{r^2 + \dots + r^2}_{16} \quad (1)$$

Our visual field used to be in 3-D space. It is difficult to understand these kinds of phenomena. Many researchers try to extend spatial index method R-tree, which is originally used in 2-D dataset [8]. When these methods proved unavailable, the term ‘‘curse of dimensionality’’ was cited [3].

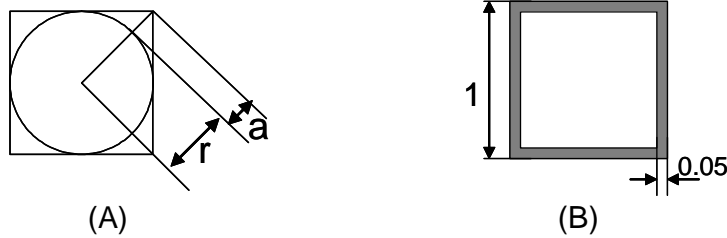


Figure 1 (a) The relation between radius and diagonal. (b) Surface and volume

2.2 Distribution of High Dimensional Dataset

To constitute the node of an index tree, the data space is partitioned. When the dimensionality increases, the exponential changes of the volume need to be considered. For example, the d -D hypercube of e edge has volume $vol = e^d$. In the case of $0 < e < 1$, the volume decreases exponentially according to d , on the other hand, when $e > 1$, the volume expands exponentially. Now we consider the distribution of data points in a hypercube. That is, how many percentage of volume is occupied by its surface? Figure 1(B) shows a square of edge 1.

The number of dimensionality	Volume of surface
2	$1 - 0.9^2 = 0.190$
3	$1 - 0.9^3 = 0.271$
.	
.	
50	$1 - 0.9^{50} = 0.994846$

Table 1 The volume of surface

The volume of the margin layer has a thickness of 0.05. Table 1 shows the volume of various aspects of dimensionality. From the table, we can find that 99.5% volume is in the surface of a 50-D hypercube. In this kind of high dimensional space, if a dataset is distributed uniformly, it can be said that *most of the data points are in the surface, not in the interior of the hypercube*. This is the motivation for this paper. To cope with this “important” surface, a surface based index structure for high dimensional space is proposed.

If the number of dimensions is d , the surface of a hypercube consists of $d-1$, $d-2$, ..., 1 , 0 -D hyperplanes. For example, a 3-D cube consists of 6 squares (2-D), 12 edges (1-D) and 8 vertices (0-D). These (hyper) planes anchor the index high dimensional data. The next Subsection explains this in detail.

2.3 Vertexes, Edges and Hyperplanes in Hypercube

A square (2-D) consists of 4 edges and 4 vertexes; a cube (3-D) consists of 6 squares, 12 edges and 8 vertexes. In general, the number of $(d-1)$ -D hyperplanes in a d -D hypercube is $2d$. We can also say the hypercube is covered by $2d$ hyperplanes. One hyperplane is also covered by consists of $(d-2)$ -D hyperplanes. Therefore, the relation of all hyperplanes is: Two $(d-1)$ -D hyperplanes intersect at a $(d-2)$ -D hyperplanes. Three $(d-1)$ -D hyperplanes intersect at a $(d-3)$ -D hyperplanes. At the end, $d-1$ $(d-1)$ -D hyperplanes intersect at a line. The number of hyperplanes which cover a hypercube is given in Lemma 1.

LEMMA 1. A d -D hypercube is covered by $0, 1, \dots, (d-1)$ -D hyperplanes, The number of i -D hyperplanes is $2^i \times C_d^i$.

Table 2 The number of hyperplanes of 20-D hypercube

The number of dimensionality of hyperplanes	Shape of hyperplanes	The number of hyperplanes
19	Hyper-plane	40
18	Hyper-plane	760
17	Hyper-plane	9,120
16	Hyper-plane	77,520
15	Hyper-plane	496,128
14	Hyper-plane	2,480,640
13	Hyper-plane	9,922,560
12	Hyper-plane	32,248,320
11	Hyper-plane	85,995,520
10	Hyper-plane	189,190,144
9	Hyper-plane	343,982,080
8	Hyper-plane	515,973,120
7	Hyper-plane	635,043,840
6	Hyper-plane	635,043,840
5	Hyper-plane	508,035,072
4	Hyper-plane	317,521,920
3	Hyper-plane	149,422,080
2	Plane	49,807,360
1	Line	10,485,760
0	Vertex	1,048,576

As dimension increases, the number of hyperplanes expands rapidly. This number is usually beyond the size of most datasets. It is therefore possible that one data point corresponds to one hyperplane of the hypercube. The hyperplane becomes the index key of a

data point. Searching similar data points becomes searching near hyperplanes. We can employ pyramid tree technique to map data point to one hyperplane; that is, partitioning the data space (or hypercube) by pyramids whose tops are the centres of hypercube and whose bases are the hyperplanes. Since most data points are in the surface of data space, the data points in the pyramid can be represented with its base (a hyperplane). If the hyperplanes are ordered in a sequence, the data points within pyramids can be indexed with linear index structure B⁺-tree. That is, a high dimensional data changes to linear data. When a range query is given, to search for similar data points becomes to find the pyramids overlapping with the query range.

3 Surface Based Spatial Index Structure

Data space is assumed to be normalized into a hypercube having edge 1. The bases of two opposite pyramids are perpendicular to an axis x_i . They can be expressed with $x_i = 0$ and $x_i = 1$. In a pyramid whose base is $x_i = 0$, every data point $(x_0, x_1, \dots, x_{d-1})$ satisfies

$$x_i \leq \min(x_j, 1 - x_j) \quad \text{where } (j = 0, 1, \dots, i - 1, i + 1, \dots, d - 1) \quad (2)$$

Based on equation (2), for a given data point, the pyramid which the data point belongs to can be determined. We use the order number of the pyramid as index key. Then B⁺-tree linear index structure is used to search for similar data.

3.1 Order of Pyramid

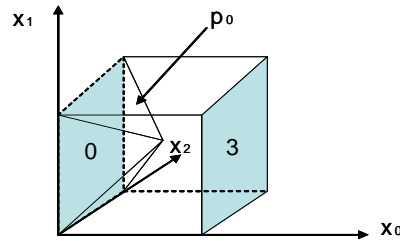


Figure 2 The order of pyramid. A 3-D cube. The bases $x_0 = 0$ and $x_0 = 1$ are assigned the order number 0 and 3 respectively. Corresponding pyramids are denoted as p_0 and p_3 .

A d -D hypercube is covered by $(d-1)$ -D hyperplanes. We assume its axes are x_0, x_1, \dots, x_{d-1} . The order numbers of the pyramids are assigned in Table 3. The order number of pyramid is determined by its base. Figure 2 shows a 3-D cube. The pyramid with base $x_0 = 0$ has an order number of 0, illustrated by p_0 . Its opposite pyramid is denoted by p_3 .

Table 3 The order of pyramids in d -D hypercube

hyperplane	order	Order of pyramids
$x_0 = 0$	0	p_0
$x_0 = 1$	d	p_d
$x_1 = 0$	1	p_1
$x_1 = 1$	d+1	p_{d+1}
$x_{d-1} = 0$	d-1	p_{d-1}
$x_{d-1} = 1$	2d-1	p_{2d-1}

3.2 Constitution of Index Key

The index keys of all data are initialized with null. For a given data point, it must belong to one pyramid. (If the data point is on the boundary of two or more pyramids, its belongingness will be discussed in Section 4). When a hypercube is partitioned, the order number of a pyramid is appended to the index key. The process is done recursively. Figure 3 shows how to constitute key in a 3-D cube. The given data point assumed be in the pyramid $x_1 = 0$. First, the index key is initialized with null. Second, since the data point belongs to the pyramid whose base is $x_1 = 0$, the order number “1” (refer the Table 3) is appended to the index key as shown in subfigure (A). To partition data space recursively, the data point is projected into the pyramid’s base as illustrated in the subfigure (B). The base is partitioned into 4 triangles. Since the data point is in No. 3 triangle (described in Table 3), the index key becomes longer by adding “3” as shown in subfigure (C). Finally, we assume the triangle is divided into 8 slices. The slice number “2” is appended into the data key. The total index key of the given data point consists of “1”, “3”, “2” as shown in subfigure (D). It can easily be combined to an integer, such as $(1*4+3)*8+2=58$. In this formula, the coefficients “4” and “8” are total numbers of lines and slices shown in the subfigure (C), (D) respectively. The index key is easy to be decoded and find which pyramids the data points belong to. In general, the partitioning process can be described with the 2 steps below.

1. A d -D hypercube is partitioned into $2d$ pyramids, their tops are the center points of the hypercube, and their bases are $(d-1)$ -D hyperplanes.
2. Within one pyramid, we projected all data into its base. Step 1 is repeated within the base of the pyramid. The base is also a hypercube. Its dimension is $(d-1)$. The base can be split into $2(d-1)$ pyramids as shown in step 1.

Along with more partition to be done, the index key becomes longer by appending the order number of its pyramid. On the last partition, the slice number which the data belongs to is appended to the index key. This algorithm is inspired by pyramid-tree technique [5]. However, pyramid-tree only partitions the hyper cube only one time.

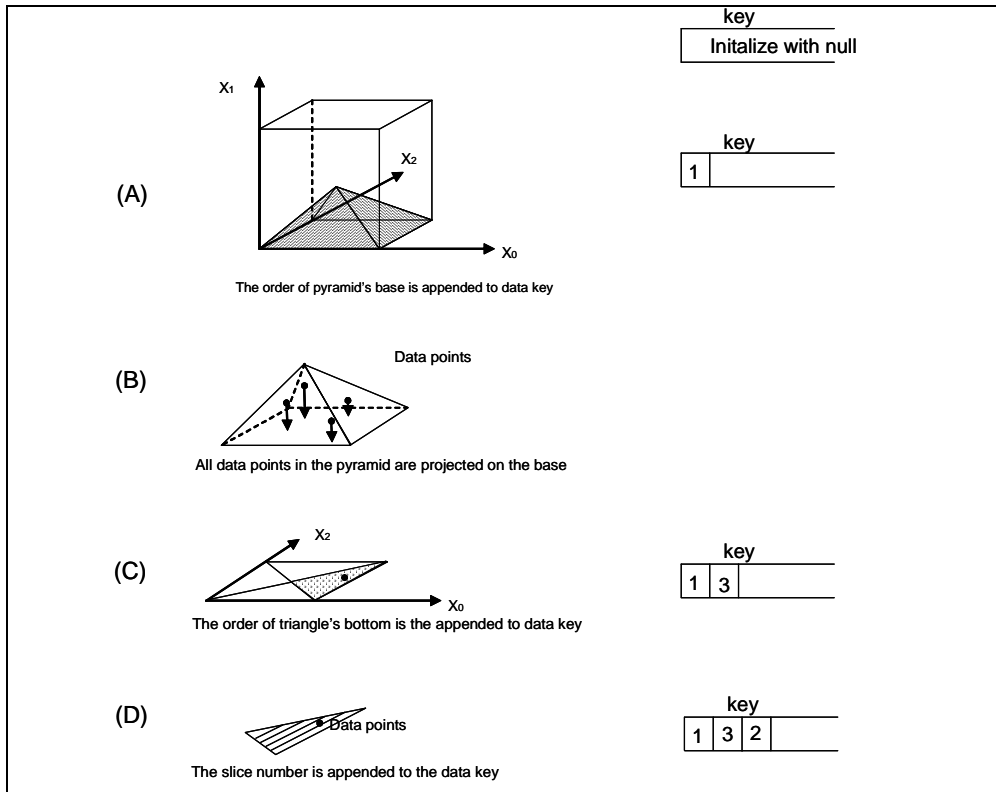


Figure 3 Construction of key. The data space is partitioned recursively, key of a data is composed the order of bases of the pyramids which the data in.

3.3 Range Search

By using the constitution of index key described in Section 3.2, the index values are lined on one sequence. We can use classical index structure B^+ -tree to do range search. Given a query range, we can calculate the pyramids overlapping with the query range. If we denote a given d -D range as $[X_{0 \min}, X_{0 \max}]$, $[X_{1 \min}, X_{1 \max}]$, \dots , the method calculating the overlapping pyramids can be described using Figure 4 as below.

Figure 4(A) shows the conditions of pyramid p_i ($0 \leq i \leq d$) which overlap the query range. The following formula describes the conditions:

$$X_{i \min} \leq X_{j \max} \quad (3)$$

$$X_{i \min} \leq 1 - X_{j \min} \quad (4)$$

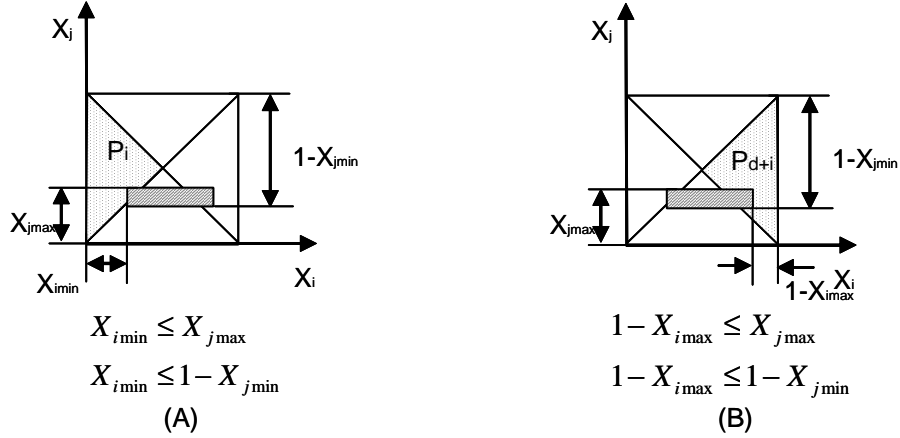


Figure 4 Finding overlapping pyramid. The pyramids overlapping with search range can be calculated with the formula shown at below.

As shown in subfigure (A) pyramid p_i overlaps with query range. The Equation 3 and 4 are satisfied. For the other side of a pyramid p_{d+i} ($0 \leq i \leq d$), as shown in Figure 4 (B). The condition of overlapping with query range can be explained using the following formula.

$$1 - X_{i\max} \leq X_{j\max} \quad (5)$$

$$1 - X_{i\max} \leq 1 - X_{j\min} \quad (6)$$

The above computing process should be done recursively just like a computing index key. All pyramids overlapping with query range will be searched by using B⁺-tree index structure. Figure 5 illustrates the pyramids (or index keys) translated from a given query range. The pyramids p_0 and p_2 do not overlap with query range. We can omit to check their sub pyramids. After two partitions, we find $p'_0 \dots p'_j p''_1 \dots p''_k$ overlap with query range. We have to make search for these index keys. The more time partition, the more index keys are needed to search. This is the reason that we can not do more partition of hyper-cube, although the accuracy of index keys approve, when more partition done.

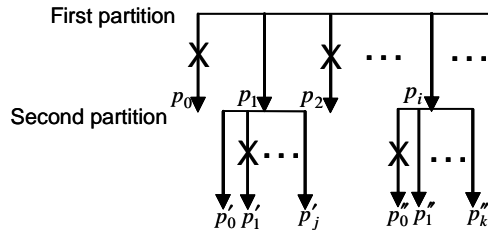


Figure 5 Translation from query range to index keys. The mark “X” means a pyramid does not overlap with query range. In the second partition, only overlapping pyramids are checked whether their sub pyramids overlap with query range or not.

4 Boundary of Pyramids Apportionment

A boundary of pyramids belongs to more than two pyramids. As an effective index structure, it is necessary to apportion one boundary to only *one* pyramid. Moreover, the boundary should be averagely apportioned. In the case of a 2-D square, there are 4 boundary lines which can be divided into 4 triangles, as shown in Figure 6. For example, the boundary line \overline{OA} is apportioned into the triangle $\triangle OAD$. \overline{OB} , \overline{OC} , \overline{OD} are apportioned into $\triangle OAB$, $\triangle OBC$, $\triangle ODC$ respectively.

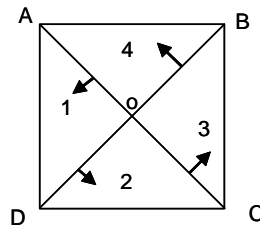


Figure 6 Boundary lines apportion for a square

In the 3-D case, there are 6 pyramids. They intersected with 12 triangle planes as shown in Figure 2. These 12 boundary planes can be divided into the 6 pyramids by 2. However, the problem is how to apportion line boundaries. There are 8 line boundaries which can not be equally divided into 6 pyramids. One reasonable apportion is distribute the 8 line boundaries 2, 2, 1, 1, 1, 1 into these pyramids.

When a data point is in a boundary of pyramids, a policy must be determined which pyramid the data point should be included in. Similarly, when the query range spans a boundary of pyramids, only one pyramid is considered. To solve the problem, a concept *radiant value* is introduced in this paper as a criterion how to divide a boundary to pyramids.

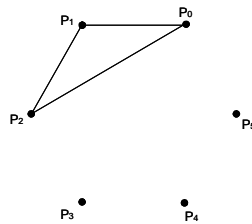


Figure 7 Boundary graph. A boundary line in a 3-D Cube

A hyperplane (or line, vertex) boundary is the intersection of more than 2 pyramids. If a data point is on a boundary, it is simple to compute which pyramids construct the boundary by using Equation 2. A graph can be constructed for a boundary as shown in Figure 7. It illustrates 6 pyramids of a cube. Every pyramid (p_0, p_1, \dots, p_5) is denoted as a point in the graph. Two pyramids which are connected by a line in Figure 7 are intersected. For example, p_1 and p_2 intersected. It is assumed that the points are put in counter clockwise order. Figure 7 shows a boundary which is constructed by three pyramids p_1, p_2, p_3 . Note that a

boundary is shown a complete sub-graph in the boundary graph. In principle, boundaries should be balanced distributed to pyramids. In other words, all pyramids are apportioned an average number of boundaries. The concept of radiant value defined by Definition 1 is used to determine a boundary belongs to which pyramid.

Definition 1 The radiant value of a vertex p is a binary number. It is initialized with null. All other points are traced in clockwise order and if a line is connected with p , then 1 is appended. Otherwise, 0 is appended.

Therefore, for the boundary in Figure 7, the point p_0 has the radiant value “000111”. The following steps express how to determine a pyramid for a boundary.

- The boundary graph is drawn and their radiant values of all points are computed.
- The data point is assigned to the pyramid having the least radiant value. If 2 or more pyramids have the same smallest radiant value, the point having the smaller subscript is chosen. For example, p_i, p_j have the same least radiant value, p_i is chosen, where $i \leq j$.

In Figure 7, point p_0 has the smallest radiant value, so the boundary intersecting three pyramids is distributed to pyramid p_0 .

5 Experimental evaluation

To evaluate the effectiveness of the new surface index structure, a collection of range queries for high dimensional dataset are performed. The surface spatial index structure is implemented based on GiST C++ Package [9] on GNU/Linux (Pentium-IV, 1GHz).

5.1 The relation between the size of candidate set and CPU time

Surface based index technique; include pyramid-tree which is a special case of our method, filters out non-related data points for a similarity query. Smaller candidate set is desirable. It is because that we have to consume I/O and CPU costs to refine every candidate to get exact answers. If the data space (or hypercube) is partitioned more times, the index keys are more accuracy and the candidate set becomes smaller. However, as mentioned Section 3.2, we have to make more queries as shown in Figure 5. That is a trade-off between accuracy of keys and query times.

We use a real dataset to show the trade-off. The real data is hue histogram exacted from color images. The dimension is 8 and the size of dataset is 100,000. Query data points were picked up from dataset randomly. The query range was set 1%. The more query range returns too more answers, analogically, too small query range returns only itself. The 1% query range returns about 10-100 answers which suggest meaningful for similarity search.

Figure 8 shows the number of candidates and CPU time according to three different partitions. If we stop at the first partition, candidate set is bigger. We have to use more CPU time to refine candidates to get real answer. If we partition hypercube by 3 times, we have a small candidate set. However, too many pyramids overlapping with query range. We

must consume more CPU time in B⁺-tree. In the result, we found two time partition is the best one for the real dataset. Its CPU time is the smallest as shown in Figure 8 (B).

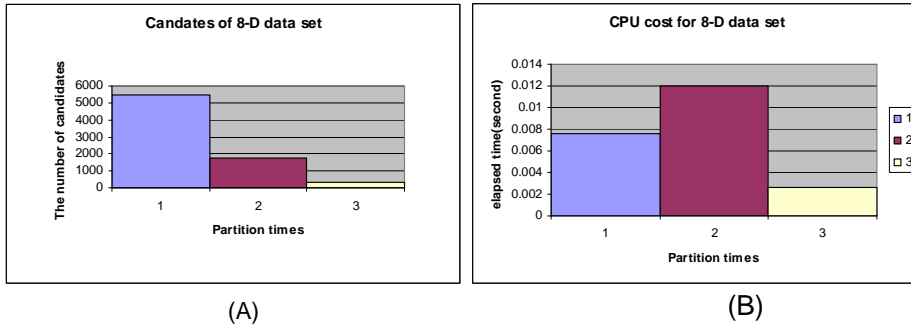


Figure 8 The number of candidates and I/O cost for real data set.

5.2 The number of page access and CUP time

To evaluate our index method on different dimensional space, we generate 15, 20, 25, ..., 85-D dataset. The data is normalized into $[0,1)$. Their sizes are 100,000. The query ranges are 2% of the data space. This query range returns properly number of answer 10-50. The node size of B⁺-tree was set at 8K bytes. Similarity search range 1000 data points were randomly selected.

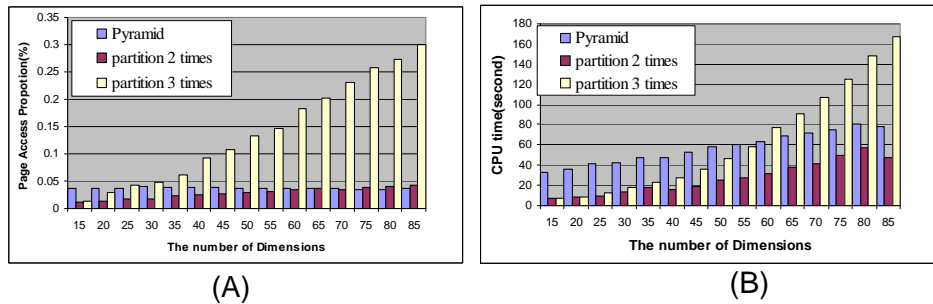


Figure 9 (A) The number of page access. (B) Range search time

Figure 9(A) shows the relationship between the number of page accesses and the number of dimensions. Note that the pyramid-tree index is a special case when the hypercube partition in "1 times", its pyramid's base is $(d-1)$ -D hyperplane. From the number of page access, we also found "partition 2 times" is the more effective than others. Figure 9(B) shows the search time in three spatial index structures. It is clear that, again, the partition-2-times method has the best performance, much better than "partition 1 time method" (i.e., the Pyramid trees). Figure 9(B) shows the search time in three spatial index structures. It is clear that, again, the "partition 2 times" is the best selection, much more effective than partition 1 time (i.e., the Pyramid trees).

6 Conclusions

In this paper, we have proposed a new index structure based on recursive partitioning space. To break the curse of dimensionality, high dimensional data points are transformed to 1-dimensional values. Therefore, classical index structures such as the B⁺-tree can be adapted. By partitioning the space recursively, our approach overcomes the restriction of 2d pyramids in the Pyramid-tree. More pyramids are partitioned and the selection of key is improved. In future work, we will estimate the optimal number of partitions required to construct an index structure considering data distribution.

7 Acknowledgments

The work reported in this paper was partially supported by the Australian Research Council's Discovery Project Grants DP0344488 and DP0345710.

References

1. An, J., Chen, H., Furuse, K., Ishikawa, M., and Ohbo, N.: The convex polyhedra technique: An index structure for high-dimensional space. Proc. of the 13th Australasian Database Conference (2002) 33-40.
2. An, J., Chen, H., Furuse, K., Ohbo, N.: CVA-file: An Index Structure for High-Dimensional Datasets, Journal of knowledge and Information Systems. to appear.
3. Beyer, K. S., Goldstein, J., Ramakrishnan, R. and Shaft, U.: When Is "Nearest Neighbor" Meaningful. When Is "Nearest Neighbor" Meaningful (1999) 217-235
4. Berchtold, S., Keim, D., Kriegel, H.-P. : The X-tree: An Index Structure for High-Dimensional Data. 22nd Conf. on Very Large Database, 1996. Bombay, India, pp. 28-39.
5. Berchtold, S., Keim, D., Kriegel, H.-P.: The pyramid-Technique: Towards Breaking the Curse of Dimensional Data Spaces. Proc. ACM SIGMOD Int. Conf. Management of Data, Seattle, 1998, pp. 142-153
6. Beckmann, N., Kriegel, P. H. Schneider, R., and Seeger, B.: The R*-tree: an efficient and robust access method for points and rectangles. Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data (1990) 322-331.
7. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. Proc. 23rd Int. Conf. on Very Large Data Bases, Athens, Greece, 1997, pp. 426-435.
8. Guttman, A.: R-tree: a dynamic index structure for spatial searching. Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data (1984) 47-57.
9. Hellerstein, J. M., Naughton, J. F., Pfeifer, A.: Generalized search trees for database systems. Proc. of the 21th VLDB conference, Zurich, Switzerland, Sept. 1995, pp. 562-573.
10. Katayama, N. and Satoh, S.: The SR-tree: An index structure for high-dimensional nearest neighbour queries. Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data (1997) 369-380.
11. Ooi, B. C., Tan, K. L. Yu, C. and Bressan S.: Indexing the Edges - A Simple and Yet Efficient Approach to High-Dimensional Indexing. PODS 2000: 166-174
12. Weber, R. Schek, J. H. and Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. Proceedings of 24th International Conference on Very Large Data Bases (1998) 194-205.
13. Zhang, R. Ooi, B. C. Tan, K. L.: Making the Pyramid Technique Robust to Query Types and Workloads. ICDE 2004: 313-324