Chapter 3

CVA-file: An Index Structure for High-Dimensional Datasets

We present the CVA-file indexing technique in this chapter. It is a new version of VA-file by applying a novel method of dimensionality reduction to VA-file.

3.1 Introduction

Constructing efficient indexing mechanisms for high dimensional data is a challenging topic because of the *dimensionality curse.* Some of recent researches conclude that it is inherently impossible to construct efficient indexing mechanisms under the assumption that data points are uniformly distributed and dimensions are not correlated[9, 11].

However, it is very unlikely that data points in real data sets are uniformly distributed and dimensions are independent of each other. We consider some real feature vector

datasets of UCI Machine Learning Repository [1] In Figure 3.1, the histogram of coordinate is illustrated (the coordinates are normalized between [0,1)). The histogram is constructed from the coordinates of all data. The vertical axis shows the percentages of number of coordinates within intervals. The intervals are assumed the same length. We can find that the length of intervals becomes shorter as the dimensionality is higher. In the other word, there are only small percentage of dimensions which have significant values in high dimensionality. For example, in 64-dimensional image color histogram feature vectors, about 78.5% of coordinates are in $(0.0, 0.05)$. Moreover, in high dimensionality, it is likely that the distribution of coordinate follows the Zipf's law[41]. We abstract this kind of data as blow:

$$\text{Entry1} \rightarrow 0, X, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, X, 0, 0, 0, 0, 0, 0, 0, 0, ..., 0, 0, X, 0$$

$$\text{Entry2} \rightarrow 0, 0, 0, 0, X, 0, 0, 0, 0, 0, 0, 0, 0, X, 0, 0, 0, 0, 0, 0, 0, 0, 0, ..., X, 0, 0, 0$$

$$\vdots$$

The notation 0 denotes that a coordinate is near 0 (e.g. $0.0003, 0.002$). While the notation X denotes that a coordinate is relatively larger than 0 (e.g. $0.15, 0.3$). The data space can be imagined as Figure 3.2. We assume a data point v has coordinates $v.x$, $v.y$, $v.z$, ... in x, y, z, ... axis. The most of data points are in *one* subspace consisting of several original axes. That is, $v.x \doteq 0$, or, $v.x \doteq v.y \doteq 0$. Although the high dimensional space is sparse as many researchers said [1, 9], in many real datasets the data concentrates in its subspaces as shown in Figure 3.2.

[1] http://www.ics.uci.edu/ ~mlearn/MLRepository.html

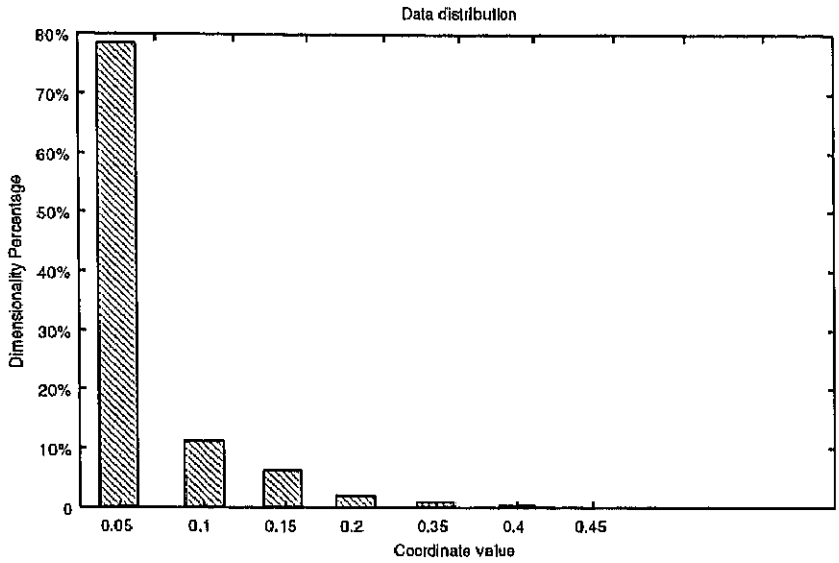


Figure 3.1: Percentage of Dimensions According to Coordinate

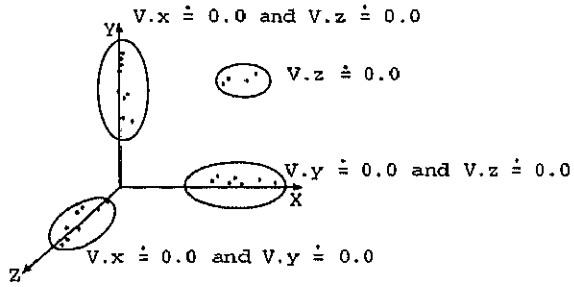


Figure 3.2: Data Distribution in High Dimensionality

Observation 1 *For many high dimensional datasets, a large portion of coordinates have relatively very small values in the scope ranging over all coordinates.*

By exploiting the non-uniformity and the correlation property of real data sets, we can have room for avoiding the dimensionality curse. Such kinds of research works include the projected clustering called PROCLUS[2]. This clustering algorithm finds subspaces in original dimensionality. One cluster corresponds to one subspace composed of a small set of dimensions. The sets of dimensions selected for each subspace need not be identical.

That is, dimensions are reduced *locally* for each cluster. This technique is also useful for constructing indexing mechanism.

For dealing with data of higher dimensionality, a commonly used approach is the dimensionality reduction. The FastMap technique proposed in [18] projects data points in a low dimensional space. In [35], another technique applying the FastMap to L_1 distance space is proposed. The Karhunen Loeve Transform (KLT)[22] is a transform by estimating the correlation matrix for the given data set, and derives the unique decorrelating transform for it. These techniques reduce the dimensionality *globally*. Therefore, all data points are represented in a common (reduced) subspace. The LDR[14] is another related work. It finds local correlations in the data and performs dimensionality reduction on the locally correlated clusters of data individually. The convex polyhedra technique[4] is another local dimensionality reduction scheme. It decomposes a data space into convex polyhedra. The dimensionality of each data point is reduced according to which polyhedron includes it.

In this thesis, we propose a more flexible dimensionality reduction in which the dimensionality is reduced in a datawise way. Its method is very simple, it reduces the coordinates whose values are smaller than a critical value. Therefore each data point is identified by remaining coordinates having significant values. In a data space, every data point is treated as a point in a subspace,which may not be identical for each data point. Moreover, the number of dimensions of each subspacemay also not be identical. From the Observation 1, our technique has the advantage that the loss of information caused by the dimensionality reduction is decreased.

The proposed technique is widely applicable to a class of indexing mechanisms. By applying it to VA-file[38], a new index structure called CVA-file is developed. The size of data points in index files is significantly reduced for skewed data.

3.2 The CVA-file technique

There are many spatial tree structures [7, 8, 26] proposed. In all these structures, tree nodes are accessed in a random way which is very ineffective. Because most accesses can be omitted during the pruning process, the effectiveness of an index is yet remarkable in low dimensionality. However, when dimensionality becomes higher, the data space can not be divided clearly into tree nodes. As a result, most of nodes are overlapping each other and the pruning power is lost. The spatial index becomes less effective than linear scanning[8, 38]. Recently, a non-tree index structure VA-file[38] was proposed. Because it does not use hierarchical tree structure, it is not hindered by the problem of overlapping. In the following, we explain the idea of CVA-file as well as its original version VA-file.

3.2.1 VA-file

VA-file[38] is an index structure which accelerates the sequential scan by the use of data compression. The k-NN search process in the VA-file scheme is divided into two phases in which two files are accessed respectively. One file is a bit compressed, quantized version of the data points, and the other is their original data file. Both files are unsorted, but the positions of the data points in the two files are identical. In the first phase the quantized

data is loaded into main memory. If the lower bound of the coming data point exceeds k-th largest upper bound of candidates, it is filtered since at least k better candidates already exist. Otherwise it is inserted into a candidate set. Intuitively, better approximation causes a tight bounds from the query point, and has only a few candidates delivered to the second phase. In the second phase, the candidates are refined by calculating the exact distance from the query point. The refinement is processed in an increasing order of the lower bound of candidates. The process is terminated when the lower bound exceeds the k-the distance in the answer set. Because random accesses occur in the second phase, the better approximation is disable. On the other hand, the number of page accesses in the second phase is proportion to the size of the approximation file.

3.2.2 VA-file in KLT domain

For evaluating the effectiveness of CVA-file, we compare it with a well-known global dimensionality reduction, called KLT (Karhunen Loeve Transform) [22]. After the transformation by using KLT, the data set becomes well scattered on several eigen vectors corresponding to the largest eigen values. Principal components of the original data vectors are preserved in these dimensions. It is also possible to combine VA-file with KLT by quantizing on the principal dimensions created as mentioned above. Then k-NN is processed as follows. In the first phase, filtering is performed on the KLT domain, and page accesses decreases because of the smaller VA-file. The exact distances between the query point and candidates are calculated in original domain in the second phase.

3.2.3 CVA-file

As a solution to the tradeoff between tightness of bounds and size of index file, we try to decrease the size of index files by using dimensionality reduction without losing the tightness of bounds. We introduce the two ways of datawise dimensionality reduction.

Dimensionality Reduction Based on Convex Polyhedra

A D-dimensional unit supercube is covered by $m(0 \leq m \leq D-1)$-dimensional hyperplanes(surfaces). For example, a cube is covered by 6 squares. It is also covered by 8 lines and 12 points. The number of dimensions for a point can be considered as 0. In general, for a d-dimensional supercube, the number of m-dimensional hyperplanes covering it is $2^{d-m} \cdot \binom{d}{d-m}$.

The Pyramid Technique[10] uses $(D-1)$-dimensional hyperplanes as bases as shown in Figure 3.4(a). A hypecube is decomposed into $2D$ pyramids. Therefore D-dimensional points can be mapped into a 1-dimensional space. The classical B^+-tree index structure can be used for similarity search. Figure 3.3 illustrates the range query q of length α in pyramids of a unit space. In this figure, if the center of the query q is placed in one of the white triangle areas, the search procedure can be localized to the triangle. That is, we can omit to check data points stored in other triangles. If data points are uniformly distributed, the probability that a query is localized to a single triangle is proportional to the total amount of the areas of triangles. However, this amount rapidly decreases as the dimensionality becomes higher. Actually, the amount of the triangles is $(1-2\alpha)^d/(2d)$,

and this amount becomes almost 0 when d is large, even if α is very small.

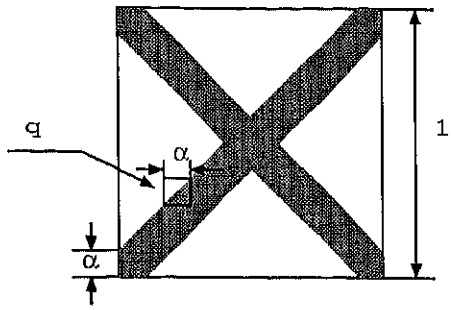


Figure 3.3: Range Query in Pyramid-tree

This is called curse of dimensionality. But pyramid technique can be extended, that is, a convex polyhedron has m-dimensional hyperplanes as bases in Figure 3.4(b). The Pyramid Technique partitions a unit supercube into $2D$ pyramids of equal size, while convex polyhedra technique partitions it into $2^{D-m} \cdot \binom{D}{D-m}$ convex polyhedra of equal size. Thus, we call our technique the *Convex Polyhedra Technique*. Of course, this technique is identical to the Pyramid Technique when $m = D - 1$.

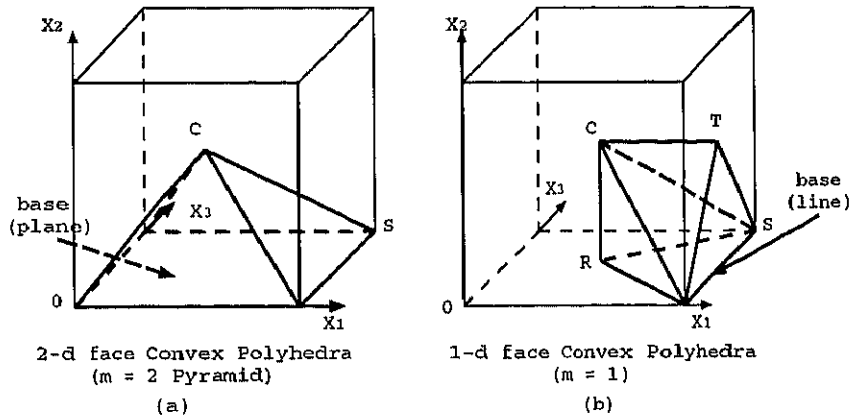


Figure 3.4: Convex Polyhedra

For a D-dimensional data set, the data space is partitioned into convex polyhedra based on $m(m \leq D)$-dimensional faces. A data point has two groups of coordinates. They have

m and $D - m$ coordinates respectively. We assume that the latter coordinates, that is, which are nearer to base than the others are, are reduced, for example, if a data point has coordinates $(x_1, x_2, x_3) = (0.02, 0.3, 0.06)$, the coordinates $x_1(0.02), x_3(0.06)$ are reduced.

Dimensionality Reduction Based on a Critical Value

For a D-dimensional normalized data set (the range of coordinate value is normalized to [0,1) in advance), a critical value $e\ (\leq 1.0)$ is determined. For each data point, the dimensions are divided into two groups as mentioned above. One contains dimensions whose coordinate values are larger than the critical value e, and the other contains the remaining dimensions. Though this thesis, dimensions of the former part is called *effective dimensions* set. The latter is called *non-effective dimensions* set. For example, if parameter $e = 0.1$ is given, then dimensions of a data point $(x_1, x_2, x_3, x_4, x_5) = (0.01, 0.25, 0.05, 0.2, 0.1)$ are divided into two groups (x_2, x_4) and (x_1, x_3, x_5), where x_2, x_4 are effective dimensions and x_1, x_3, x_5 are non-effective dimensions with respect to the data point. Intuitively, the effective(or non-effective) dimensions might be different for different data points, i.e. though dimension x_1 is an effective dimension for this data point, x_1 might be a non-effective dimension to another data point.

Comparing to the dimensionality reduction based on convex polyhedra, this technique of dimensionality reduction is more flexible and less lossy in distance information.

Now we apply the dimensionality reduction based on critical value to VA-file. Assume that b_d $(i = 1, 2, \ldots, D)$ bits are used for approximating data points for each dimension. CVA-file is constructed by appending the entries of data points one by one. The entry of a

data point v is as follows.

1. Add a D-bit (D is the number of dimensions for dataset) header to the entry. All header bits are initialized to 0.

2. For each dimension, if its coordinate value is larger than the critical value e, then append the approximation of this coordinate value, and set the corresponding bit of the header to 1. For an effective dimension x_i, its approximation is $\lfloor x_i \times 2^{b_i} \rfloor$.

Figure 3.5 shows the structures of VA-file and CVA-file. In CVA-file of Figure 3.5(b), header fields "dim. inf." preserve the information of effective dimensions for each data point, and the "VA-data" fields preserve approximation of coordinate values of the effective dimensions. Although the header fields are overheads, the length for storing one data point in CVA-file is much shorter than that in VA-file due to omittance of some coordinates. Hence, the size of CVA-file is smaller than VA-file because data approximation of many non-effective dimensions are omitted.

As an example, let $d = 4$, $e = 0.2$, $b_2 = 3$, $b_3 = 2$, and $v = (0.1, 0.3, 0.6, 0.2)$. Then the index entry for v is $(0110, 010, 10)_2$. Because the second and third dimensions are the effective dimensions, the header becomes "0110" by setting the corresponding bits, the approximation of coordinate values, 0.3 ($\rightarrow \lfloor 0.3 \times 8 \rfloor = 2$), falls into the third partition of the $8(= 2^{b_2})$ partitions counted from "000", and so forth.

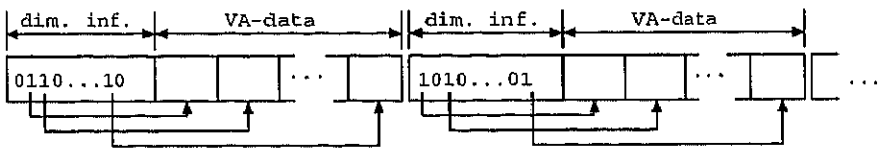
Algorithm 1 shows algorithm creating an index file in CVA-file method. In a D-dimensional dataset $\mathcal{O}$, a data point is denoted by v_i, its coordinates are presented with $v_i.x_d$ and approximation is $r_i.x_d$.

Input:
Dataset: $\mathcal{O}$
Number of dimension: D
Number of bits for each dimension $x_d (1 \leq d \leq D)$: b_d
Create CVA-file($\mathcal{O}$, d, $b_1, ..., b_D$)

begin
foreach dimension $x_d (d = 1 : D)$
{calculate length of dimension $I_d (1 \leq d \leq D)$}
$I_d = \frac{1}{2^{b_d}}$
endforeach
foreach data point $v_i \in \mathcal{O}$
$header \leftarrow d$Bits, $data \leftarrow \emptyset$
{D bits header is prepared, and $cva\ data$ is initialized}
foreach dimension $x_d (d = 1 : D)$
if($v_i.x_d > e$)
$header[i] \leftarrow$ 1b {set 1 in corresponding bit of the header}
$r_i.x_d = \lfloor \frac{v_i.x_d}{I_d} \rfloor$
$data \leftarrow data \cup r_i.x_k$
{append quantized number to the end of data}
endif
endforeach
writefile(cva_file, $header \cup data$)
{write this entry to the cva file}
endforeach
end

Algorithm 1: Create the CVA-file which consists of approximations of all data points.

(a) VA-file Structure



(b) CVA-file Structure

Figure 3.5: CVA-file Structure

Table 3.1: Notations and Basic Definitions

e	critical value of coordinate
D	number of dimensions
N	number of data points
i	range over data point, $i \in \{1, \ldots, N\}$
v_i	i-th data point
b	number of bits per approximation
I_d	length of region in $x_d (1 \leq d \leq D)$ dimension
q	a query
l_i, u_i	bounds: $l_i \leq L_p(q, v_i) \leq u_i$
$v_i.x$	v_i's coordinate of x dimension
b_d	bits per approximation in dimension x_d
$r_i.x$	region No. where v_i falls in dimension x
n	number of data points in result set
L_p	distance function $L_p(q, v_i)$
$l_i.x, u_i.x$	contribution to l_i,u_i for dimension x

3.2.4 Estimating Bounds in CVA-file

Assume the critical value of coordinate to be e, let N be the number of data points, and D be the number of dimensions. We use $i \in \{1, \ldots, N\}$ to range over data points, and the dimensions are divided into effective dimension set X_i, and non-effective dimension set X_i'. v_i denotes an individual data point, and $v_i.x_d$ is a coordinate of x_d dimension. b is the number of bits required in each approximation, and b_d indicates number of the bits assigned to dimension x_d. All notations are summarized in Table 3.1.

Figure 3.6 illustrates the notation. Assume that data point v_i is in the 3rd region of dimension x_d, The region number is counted from 0 so we have $r_i.x_d = 2$. The $r_i.x_d$ can be calculated by the equation $r_i.x_d = \lfloor \frac{v_i.x_d}{I_d} \rfloor$. The notation $v_i.x_d$ is the x_d's coordinate. I_d is the length of a region(shaded) in dimension x_d. Intuitively the equation $I_d = \frac{1}{2^{b_d}}$ holds.

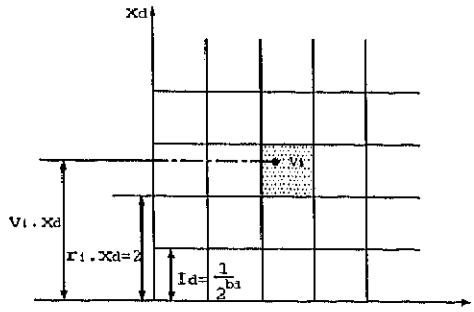


Figure 3.6: Notation.

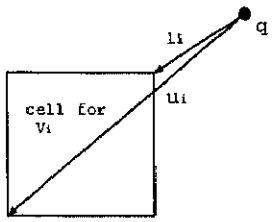


Figure 3.7: Lower Bound and Upper Bound in VA-file

Let us consider a query q and a distance function L_p, for some p. An approximation of v_i determines a lower bound l_i and an upper bound u_i such that:

$$l_i \leq L_p(q,\ v_i) \leq u_i$$

This is sketched in Figure 3.7. The lower bound l_i is simply the shortest distance from the query to a point in that cell. Analogously, the upper bound u_i is the longest distance to a point in that cell. Formally, l_i and u_i are derived as follows. The bounds l_i and u_i are total of all effective dimensional bounds $l_i.x_d\ u_i.x_d(x_d \in X)$ and non-effective dimensional bounds $l_i.x_d\ u_i.x_d(x_d \in X')$ as shown in Figure 3.8:

$l_i.x_d\ u_i.x_d(x \in X)$ are the lower and the upper bounds of effective dimensions, respectively. They are computed like in VA-file as Figure 3.9.

$$l_i = \left(\sum_{x_d \in X_i} l_i.x_d{}^p + \sum_{x_d \in X_i'} l_i.x_d{}^p \right)^{\frac{1}{p}} \tag{3.1}$$

$$u_i = \left(\sum_{x_d \in X_i} u_i.x_d{}^p + \sum_{x_d \in X_i'} u_i.x_d{}^p \right)^{\frac{1}{p}} \tag{3.2}$$

Figure 3.8: Lower and upper bound.

$$l_i.x_d = \begin{cases} q.x_d - I_d\ (r_i.x_d + 1) & (r_i.x_d < r_q.x_d) \\ 0 & (r_i.x_d = r_q.x_d) \\ I_d\ r_i.x_d - v_q.x_d & (r_i.x_d > r_q.x_d) \end{cases}$$

$$u_i.x_d = \begin{cases} q.x_d - I_d\ r_i.x_d & (r_i.x_d < r_q.x_d) \\ max(q.x_d - I_d\ r_i.x_d,\ I_d\ (r_i.x_d + 1) - q.x_d) & (r_i.x_d = r_q.x_d) \\ I_d\ (r_i.x_d + 1) - q.x_d & (r_i.x_d > r_q.x_d) \end{cases}$$

Figure 3.9: Lower and upper bound for effective dimension($x_d \in X_i$).

As with data points, a query q consists of components $q.x_d$, and these fall into regions numbered $r_q.x_d$. For effective dimensions, the components $r_i.x_d$ $(x_d in X_i)$ of v_i are stored in CVA-file. For non-effective dimensions, $r_i.x_d$; $(x_d in X_i')$ are not stored in CVA-file, they are needed to be estimated.

We consider dimension x_1 in Figure 3.10(A). According to a critical value, a data point o_1 has coordinate of x_1 in shaded portion, its coordinate of x_1 is omitted, i.e. x_1 is a non-effective dimension with respect to data point o_1. On the other hand, x_2 is an effective dimension for data point o_2, because it has coordinate of x_1 over critical value. The converse idea can estimate the range of coordinate of non-effective dimension, that is, if the coordinate of x_1 of a data point is omitted, its range of x_1 is in the shaded portion as shown in Figure 3.10(A).

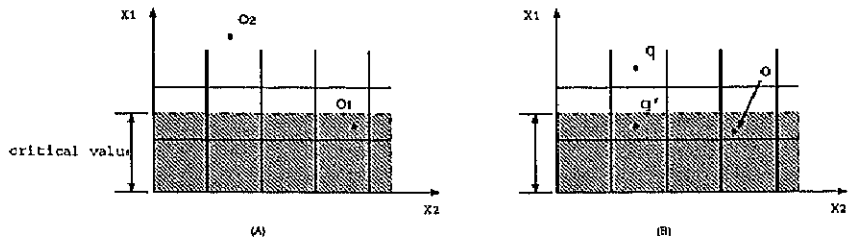


Figure 3.10: Lower Bound and Upper Bound of Non-effective Dimension

$$l_i.x_d = \begin{cases} 0 & (q.x_d < e) \\ q.x_d - e & (otherwise) \end{cases}$$

$$u_i.x_d = \begin{cases} max(e - q.x_d,\ q.x_d) & (q.x_d < e) \\ q.x_d & (otherwise) \end{cases}$$

Figure 3.11: Lower and upper bound for non-effective dimension($x_d \in X_i'$).

Assume that data point o have non-effective dimension x_1 as shown in Figure 3.10(B). Let us calculate the lower and upper bound between query point q and o. The x_1's lower and upper bounds of o are the distances from q to the shaded portion. Consequently, we should consider two cases according to dimension x_1's location of q as shown in Figure 3.11.

3.2.5 CVA-file Algorithm

CVA-file algorithm has two phases(filtering and refinement) as VA-file does. In the first phase, the approximations of data points are scanned linearly. The lower and upper bounds are calculated in accordance with Equation 3.1. Because some dimensions are reduced, the approximations for those dimensions have to be estimated in the way shown in Figure 3.10. Then the candidates are obtained by filtering out the most of data points. The second phase

visits data points in the original data file based on the result of the first phase. Thereby the final answer set are obtained. Note, these candidates are then visited in increasing order of their lower bounds of the distance to the query point q. The CVA-file method stops when the next lower bound exceeds k-th nearest distance obtained so far. The algorithm is descried in pseudo code in Algorithm 2.

3.3 Performance Evaluation

In order to demonstrate that CVA-file method is an effective alternative of VA-file of skewed datasets where search is meaningful[11]. We evaluated the performance on synthetic as well as real datasets. The VA-file, SR-tree and a simple sequential scan search structures were evaluated. The synthetic dataset is generated under the assumption of Zipf's distributed of data points. We selected color histogram provided online at the UCI KDD Archive web site[2] as the real dataset.

CVA-file method distinguishes non-effective dimensions by checking the corresponding bits in a header. A little more CPU time is needed in this checking process. However, the comparing time of coordiantes between query point and approximations are decreased due to omitted coordinates. As a result, there seems little significant difference in CPU times between CVA-file and VA-file as shown in Figure3.12. We evaluated the performance based on the number of page accesses.

Let N be the size of the D-dimensional data set, and $b_d (1 \leq d \leq D)$ be the number of

[2]http://corel.digitalriver.com/

Input:
query point q,
number of nearest neighbor k
approximation file $cva\text{-}file$
data file $data\text{-}file$
Output:
k-nearest neighbors ans.
Global variables:
$QueueSortOnUpperBound[\cdot]$, $QueueSortOnLowerBound[\cdot]$.
CVA k-NN Search(q, k, $cva\text{-}file$, $data\text{-}file$)

begin
foreach approximation α_i in $cva\text{-}file$
{Phase I: candidates are obtain by calculate bound}
$l_i, u_i \leftarrow$getBound(α_i, q) {using Eq. 3.1}
if $l_i < QueueSortOnUpperBound[k]$
$QueueSortOnUpperBound[k] \leftarrow u_i$
sortQueue($QueueSortOnUpperBound$)
InsertQueue($QueueSortOnLowerBound$, l_i, i)
endif
endforeach
foreach answer $ans[i](i = 1 : k)$
{initialize the answer variable ans}
$ans[i].dst \leftarrow \infty$
endforeach
$l_i, i \leftarrow popQueue(QueueSortOnLowerBound)$
while $l_i < ans.dst[k]$
{Phase II: determine the final answer by calculating exact distance to query}
$v_i \leftarrow readfile(data\text{-}file, i)$
$ans[k].dst \leftarrow dist(v_i.q)$
sortOnDst(ans)
$l_i, i \leftarrow popQueue(QueueSortOnLowerBound)$
endwhile
end

Algorithm 2: k-NN search with CVA-file index structure. It consists of two phases(filtering and refinement).

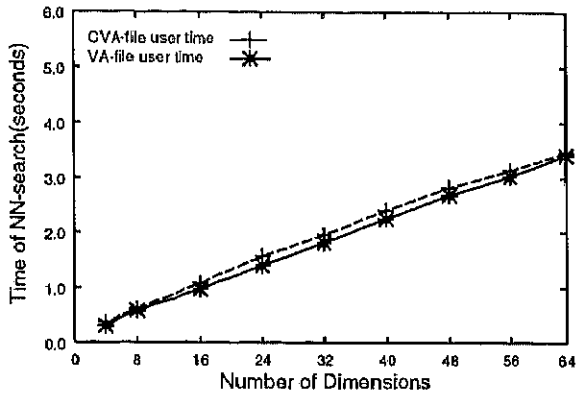


Figure 3.12: Comparison of CPU time.

bits of one dimension of an index entry. Let $\bar{b}$ be average of b_d. Then the size of VA-file is $\bar{b}dN$. While, in CVA-file, the number of bits b'_d are assigned to each dimension. Let $\bar{m}$ be the average number of effective dimensions and let the average of b'_d be $\bar{b'}$, then the size of CVA-file is $(\bar{m}\bar{b'} + d)N$. This is smaller than the size of VA-file when

$$(\bar{m}\bar{b'} + d)N/\bar{b}dN = (\bar{m}\bar{b'} + d)/\bar{b}d$$

In the simplest case in which we assign to each dimension the same number of bits as in VA-file, namely $\bar{b'} \doteq \bar{b}$, the above expression is simplified to

$$\begin{aligned}(\bar{m}\bar{b'} + d)N/\bar{b}dN &= (\bar{m}\bar{b'} + d)/\bar{b}d \\ &= \bar{m}/d + 1/\bar{b}\end{aligned}$$

Since $\bar{b}$ is about 7 or 8 [38], that is, $1/\bar{b} \doteq 0$, the size of CVA-file is smaller than VA-file at the rate of m/d. Since the number of page accesses in the phase one is in proportion to the

size of VA-file and CVA-file, CVA-file index mechanism vastly cuts down the number of page accesses in phase one. This compensates excess page accesses in phase two because of less tight bounds than VA-file.

Average number of effective dimensions m is determined by parameter e. When e becomes smaller, the number of effective dimensions m is needed to be stored in index file. When e is set to be the length of grid, the quality of bounds can be considered the same as VA-file, the number of page accesses in the first phase is saved. Consequently, no matter how far scatter the datasets are scattered, CVA-file method is never worse than VA-file method as described in [16].

3.3.1 Real Dataset

We evaluated the performance on the real dataset of color images available from the Corel Database Color Database (http://corel.digitalriver.com/) and color histograms provided online at the UCI KDD Archive web site (http://kdd. ics.uci.edu/databases/CorelFeatures). The size of the dataset is $70,000$. $4, 8, 16, \ldots, 64$-dimensional data points are extracted from the dataset. The distance function is based on the L_2 metric (Euclidean distance). and page size of index files is $8KB$.

As mentioned above, VA-file method scans the index of all data points to filter out candidates in the first phase in which data are accessed in the sequential way. This contributes to gain the performance because the cost of sequential access is significantly lower than that of random access which is used in the second phase. A factor of 5, that means

Table 3.2: Number of Page Accesses in Phase 2

Number of Dimensions	4	8	16	32	40	48	56	64
Number of Page Access	14	18	21	22	19	23	23	26

that random access is as five times slower as sequential access, is assumed in VA-file [38]. Having no hint on this factor, yet to compare with VA-file and KLT[22] approaches fairly, we evaluated the performance in other ways.

Because the factor of costs between random accesses and sequential accesses is not stable, we considered two extreme situations.

Firstly, we controlled CVA-file to have the same number of page accesses in phase two (as shown in Table 3.2) for all three index structures (CVA-file, VA-file and KLT) by tuning the parameter e, and compared numbers of sequentially accessed pages in the first phase. As shown in Figure 3.13, CVA-file outperforms VA-file and KLT. The effectiveness becomes significant as the dimensionality increases. For example, in the case of 64-dimensions, CVA-file needs a bit more than 200 page accesses, while KLT needs more than 350 and VA-file needs even more. Meanwhile, Figure 3.14 shows the number of effective dimensions corresponding to the previous figure.

Secondly, as shown in Table 3.2, the number of page accesses in the second phase are small (26 of less page accesses for 10-NN query). This tells us that the comparison of total number of page accesses within two phase is meaningful, even if the factor 5 or 10 as mentioned above is taken into consideration. Figure 3.15 shows total number of page accesses among SR-tree, VA-file and CVA-file in 10-NN query in various dimensionality.

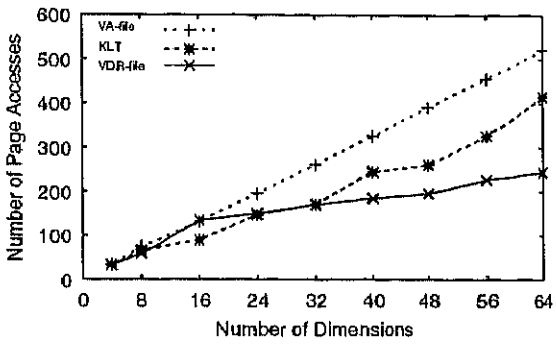


Figure 3.13: Comparison of number of page accesses of the first phase in case of having the same number of paper access in the second phase as shown in Table 3.2.

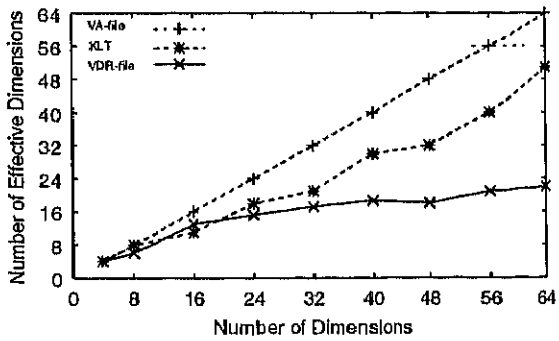


Figure 3.14: The Effectiveness of Dimensionality Reduction

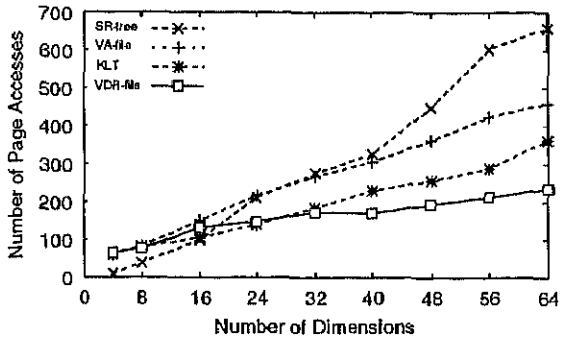


Figure 3.15: Comparison on four index structure($factor = 10$)

For each dimensionality, the number of bits assigned to each dimension for VA-file is chosen to 8 bits for 24 or less dimensionality and 7 bits for 32 or more dimensionality. These decisions are made according to [38] which concludes that this choice gives them the best approximation file.

Our experiments measured the number of the most effective dimensions of CVA-file structure. The average of the number of effective dimensions is shown in Figure 3.16. The results indicates that there is a slight increase of the number of effective dimensions in high dimensionality. Comparing to VA-file which is linear to dimensionality of the dataset, our approach reduces the dimensionality to the number of effective dimensions, which is significantly smaller than the original dimensionality. Furthermore, the effect of the reduction becomes more remarkable as the dimensionality becomes higher.

Figure 3.15 shows the number of page accesses in various dimensions. We can observe that CVA-file always outperforms VA-files, and it outperforms SR-tree and KLT for high-dimensional dataset. The margin increases as the number of dimensionality increases. For the case of 64-dimension, CVA-file cuts down 2/3 of page accesses from SR-tree ([26]),

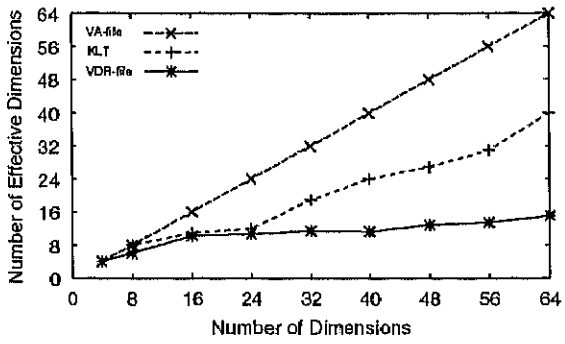


Figure 3.16: The Effect of Dimensionality Reduction (total $factor = 1$)

$1/2$ of pages accesses from VA-file, and $2/3$ of pages accesses from VA-file in KLT domain.

3.3.2 Synthetic Dataset

As the dimensionality becomes high, the coordinates distribution has the tendency of Zipf's power as seen in Section 3.1. For creating dataset based on Zipf distribution, we use random data for the coordinates which range over n partitions: $[0.0, p_1), [p_1, p_2), ..., [p_{n-1}, 1.0)$. The number of coordinates for each partition follows

$$P[X > x] \propto x^{-k}$$

where x is a lower bound of partition, and k is a coefficient of Zipf distribution. The bigger k becomes, the more the number of coordinates having value nearly 0 becomes. We assume $k = 2.5$ and the coordinates percentage is shown Figure 3.17.

The number of dimensionality ranges over $8, 16, ..., 64$. The coordinates range over $[0, 1)$. We have seen in Section3.1, the length of partitions becomes smaller when dimen-

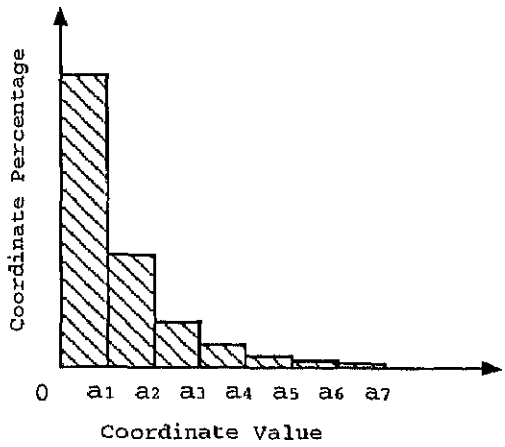


Figure 3.17: Coordinate distribution following the Zipf's power

sionality becomes higher. Thus the length of partition is assumed as follows.

$$length = \begin{cases} 0.01 & dim = 8, 16, 24, 32 \\ 0.005 & dim = 40, 48, 56, 64 \end{cases}$$

The number of synthetic data is $100,000$ for all dimensional datasets. We assume that CVA-file, VA-file/KLT domain and VA-file have the same number of page accesses in the second phase. The number of page accesses in the first phase is shown in Figure 3.18. The number of effective dimensions in CVA-file and KLT domain are shown in Figure 3.19.

3.4 Conclusions

We observed that Zipf distribution also observed in high dimensional real data. That is, there are few dimensions whose coordinate values are large, and the most of coordinate values are nearly zero. We proposed a data-wise dimensionality reduction technique, in

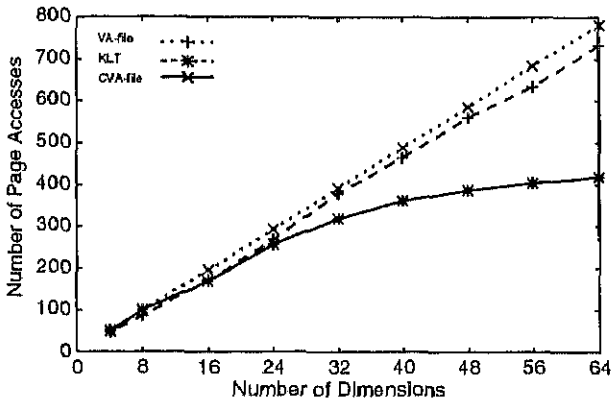


Figure 3.18: Number of page accesses in the first phase(synthetic data)

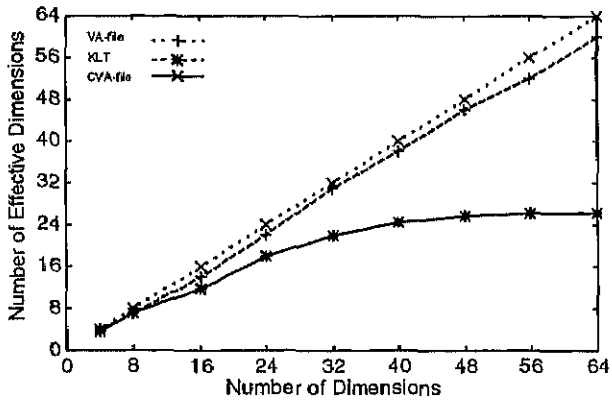


Figure 3.19: Number of effective dimensions(synthetic data)

which less of information loss is needed by dimensionality reduction. An application of our technique to VA-file implements another index file CVA-file. The performance evaluation shows significant improvements on the I/O cost of queries over original VA-file and VA-file in KLT domain for several real datasets.

In the next chapter, we present a new dimensionality reduction technique, called Datawise Dimensionality Reduction(DDR), for time series data. This is to verify that technique of dimensionality reduction datawise can be applied to various high dimensionality datasets. We show how DDR can be indexed using a multidimensional index structure. Such an index can significantly reduce the I/O accesses for similarity searching in time series dataset.