A Distance–Based Packing Method for High Dimensional Data

Tae-Wan Kim **Ki-Joune Li**

Department of Computer Science
Pusan National University
San–30, Jang–Jun Dong, Kum–Jung Gu
Pusan, Korea, 609–390
Email: {twkim,lik}@quantos.cs.pusan.ac.kr

Abstract

Minkowski-sum cost model indicates that balanced data partitioning is not beneficial for high dimensional data. Thus we study several unbalanced partitioning methods and propose cost models for them based on Minkowski-sum cost model. Our cost models indicate that the distance to one of both ends of data space dominates the expected value under uniform data distribution. We generalize studied methods to adapt to data distribution and propose a new partitioning method, called *DD–CSP* (Distance-based Distribution–adaptive Cyclic Sliced Partition), for high–dimensional index structures. At each partition, it splits data from lower end or higher end to the center of data space based on distance cost function. Based on this fact, we propose a data structure called DSR(Dimension–independent Single value Representation) which takes constant amount of storage to represent MBHs(Minimum Bounding Hyper–cubes) independent of dimension.

In our experimental studies, we compare DD–CSP with *R–tree*, *HP*, *STR*, *TGS*, and methods analyzed in our paper on real and synthetic data sets with wide ranges of dimensions and of selectivities varying from 10^{-1} to 10^{-9}. In all experiments, we show that our method, DD–CSP, outperforms all other methods and achieves up to 567% savings in response time. Thus it is a clearly winning strategy in terms of range queries and storage requirements.

Keywords: High–dimensional Space, Cost Model, Bulking–Loading, Packing, Access Method

1 Introduction

During recent years, new database applications have been studied which are basically different in many aspects to conventional database applications. These are data warehousing, multimedia database systems, etc. The common characteristics of data in these new applications are their huge size and multi-attributes. Multi-attributed data objects are transformed to multi-dimensional points called feature vectors. Similarity and range searches are one of basic functionalities required in these applications. Subsequently, the high dimensional index structure is required to process search efficiently. Packing or bulk–loading is one of the possible solutions to enhance query performance among many database techniques and is the area of our interest.

Packing methods so far proposed have two major objectives: fast index construction, fast query process. In (Bercken et al. 1997, Arge et al. 1999), they proposed dynamic bulk–loading algorithms based on buffer tree(Arge 1996) to build indices fast. Except these, most of studies have focused on fast query process and been done based on sort-partition heuristic approaches (Roussopoulos et al. 1985, Leutenegger et al. 1997, Kamel et al. 1993, Garcia et al. 1997, Berchtold et al. 1998, White et al. 1996, Gavrila 1996) on R–tree. Among them, (Berchtold et al. 1998) uses $O(N)$ split method to partition a data set instead of sorting a whole data set. Packing methods for quadtrees and gridfiles are proposed in (Hjaltason et al. 1997, Hjaltason et al. 1999) and (Leutenegger et al. 1994), respectively. In (Gavrila 1996, Pagel et al. 1995), they apply optimization techniques as the post-processing step to pack nodes. Note that NP–hardness of packing is proved in (Pagel et al. 1995).

Packing, in general, consists of two orthogonal processes as sub-modules: data partitioning and index construction. In data partitioning process, a data set is partitioned into subsets whose cardinality is at maximum a certain number of data objects, called $fanout$. Data objects are sorted according to particular methods and are assigned linearized orders. For example, (Kamel et al. 1993) assigns Hilbert orders and (Leutenegger et al. 1997) assigns tile orders. Based on these orders, data objects are partitioned. In index construction process, minimum bounding rectangles(MBR) or MBH are constructed from partitioned subsets and inserted as entries to intermediate or terminal nodes of the index structure. Top-down approach builds intermediate nodes first, whereas bottoms-up builds terminal nodes first.

For low dimensional data, it is query-efficient to minimize the perimeter of bounding boxes or to generate square-like bounding boxes(Beckmann et al. 1990, Kamel et al. 1993, Pagel et al. 1993) in data partitioning process. But for high dimensional data, it is not true (Berchtold et al. 1998). In (Berchtold et al. 1998), they showed that a heavily unbalanced split strategy, such as a 9:1 proportion results in better query performance than a balanced split strategy such as a 1:1 proportion. The branching effect of intermediate nodes of tree–like index structures becomes insignificant as dimension grows due to limited number of partitions and high overlapping volume(Berchtold et al. 1996, Katayama et al. 1997). Thus the direction of index construction does not influence critically on query performance.

The unbalanced partitioning strategy is a quite promising heuristics for high dimensional data. But, methods so far proposed do not explicitly suggest a solid metric for data partitioning. In our work, we focus on this fact and suggest a dominating metric for partition by analyzing several unbalanced data partitioning methods. Based on our analysis, we generalize studied methods to adapt to data distribution and propose a new partitioning method, called *DD–CSP*, for high–dimensional index structures. It partitions data from lower end or higher end to the center of data space based on distance cost function. At each partition, the split value at i^{th} dimension is the maximum among all *MBH*s which are partitioned from lower end or is the minimum among all MBHs which are parti-

tioned from higher end. Based on this fact, we propose a data structure called DSR associated to DD–CSP. It takes constant amount of storage to represent a MBH independent of dimension, that is $c\frac{datasize}{fanout}$ where c $\approx$ 12Bytes. Its storage gains become larger as dimension d of data grows compared to the traditional dimension–dependent MBH–representations which take about $8d\frac{datasize}{fanout}$. More details about data structures for high dimensional index, the reader may refer (Böhm et al. 2001).

In section 2, we analyze the Minkowski-sum cost model for high dimensional data suggested in(Berchtold et al. 1998, Leutenegger et al. 1998). In section 3, we provide analysis of various unbalanced partitioning strategies under uniform data distribution assumption. From the analytical results, we provide a dominating metric in the selection of split dimension. We also show the validity of our analysis of unbalanced strategies by simulations and experiments. In section 4, we propose our partitioning method, DD–CSP, and an associated data structure. In section 5, we show our experimental studies on various strategies on both real and synthetic data sets with wide ranges of dimensions and of selectivities varying from 10^{-1} to 10^{-9}. In section 6, we conclude our study.

2 Analysis of Minkowski cost model

It is known that the most important decisions in partitioning are the selection of partition dimension and the number of partitions at chosen dimension. For low dimensional data, it is query-efficient to select a dimension and number that partitions a data set into subsets equally sized to all dimensions. Thus, equally sized grid-like tiling or *equi_depth* is a good decision for low dimensional data. But, for high dimensional data, decision criteria used for low dimensional data are not useful anymore. For example, we are given an uniformly distributed data set whose dimension and size are 10 and 25600, respectively. Assume that *fanout* is 25 and we partition the data set by the *equi_depth* strategy. Then the *equi_depth* generates 1024 pages by splitting once at each dimension. Suppose a hyper cubic query with selectivity, s, 0.1% is given, its one dimensional length is $0.501(= \sqrt[10]{0.001})$. Since the side length of a range query is larger than 0.5, it intersects all MBHs generated by the *equi_depth* partition strategy. Analytically, *equi_depth* partition strategy has similar query performance to linear scan for high dimensional data even with low selectivity. This effect can be modelled more accurately by means of the analytical model, Minkowski-sum cost model, proposed in (Berchtold et al. 1998, Leutenegger et al. 1998). The expected value $E(M, \bar{\mathbf{q}})$ for page accesses upon processing a hyper cubic query $\bar{\mathbf{q}}$ with a side length q on a set M of MBHs is:

$$E(M, \bar{\mathbf{q}}) = \sum_{i=1}^{m} \prod_{j=0}^{d-1} \frac{min(high_{i,j}, 1-\mathrm{q}) - max(low_{i,j} - \mathrm{q}, 0)}{1-\mathrm{q}} \quad (1)$$

M	:	the set of MBHs where $|M| = $ m.
$high_{i,j}$	:	the rightmost value of an interval of i^{th} MBH in j^{th} dim
$low_{i,j}$	:	the leftmost value of an interval of i^{th} MBH in j^{th} dim

According to equation (1), the expected value of above *equi_depth* strategy is:

$$E_{equi\_depth}(M, \bar{\mathbf{q}}) = \frac{0.5^d}{(1-\mathrm{q})^d} m = \frac{1}{(1-\mathrm{q})^d} \quad (2)$$

where $m = 2^d$.

When q equals to 0.5, $E_{equi\_depth}(M, \bar{\mathbf{q}})$ is $|M|(= 2^d)$. Thus when q $\geq$ 0.5, a range query accesses all MBHs generated by *equi_depth* partition strategy. As a result, the balanced strategy such as *equi_depth* strategy is not proper strategy for high dimensional data.

Now we look closer at equation (1) and examine the behavior of it for unbalanced partitions. Equation (1) is expressed as the sum of the product of the one dimensional lengths of MBHs. And the contributions of the lengths of MBHs to the expected value $E(M, \bar{\mathbf{q}})$ are different depending on the intervals partitioned by a query side length q. The following example shows more specifically.

Example 2.1. *Assume that the data space is one dimension and it is equally partitioned into P_1(=7) intervals. Figure 1 shows seven partitioned intervals.*

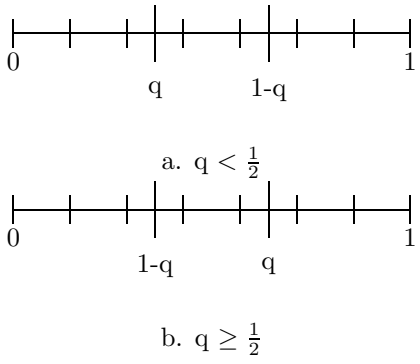


Figure 1: Intervals partitioned by q

When q < 0.5, there are six intervals intersecting [0, q] *and* [1−q, 1]. *By equation (1), the sum of the expected values of these intervals is* $2\frac{1+2+3}{P_1}/(1-\mathrm{q})$. *The expected value of each interval in* [q, 1−q] *is the same and is* $(P_1 - 6)(\frac{1}{P_1} + \mathrm{q})/(1-\mathrm{q})$. *When* q $\geq$ 0.5, *the number of intervals contained in* [0, 1−q] *and* [q, 1] *is four. The sum of the expected values of these intervals is* $2\frac{1+2}{P_1}/(1-\mathrm{q})$. *The expected value of each interval in* [1−q, q] *is the same and is* $(P_1 - 4)(1-\mathrm{q})/(1-\mathrm{q}) = P_1 - 4$. *When* q *equals to 0.3 and 0.7, the expected value is 3.08 and 5.85, respectively.*

From the example 2.1, we observe that (1) the number of intervals in the middle is directly related to the expected value. (2)The expected value of the intervals in the both ends of a dimension is directly related to the cumulative sum of lengths. As a result, we have the following proposition.

Proposition 2.1. *Suppose that a d–dimensional hypercubic MBH and a query side length* q $\geq$ *0.5 are given. Let the i^{th} dimensional length of a MBH be an interval $I = [low_i, high_i]$. Then its contribution to the expected value is*

1. $\frac{high_i}{1-\mathrm{q}}$, when $high_i < 1 - \mathrm{q}$.
2. $\frac{1-low_i}{1-\mathrm{q}}$, when $low_i >$ q.
3. 1, when $(high_i - low_i) \geq 1 - \mathrm{q}$
4. 1, when $1 - \mathrm{q} \leq high_i \leq$ q or $1 - \mathrm{q} \leq low_i \leq$ q

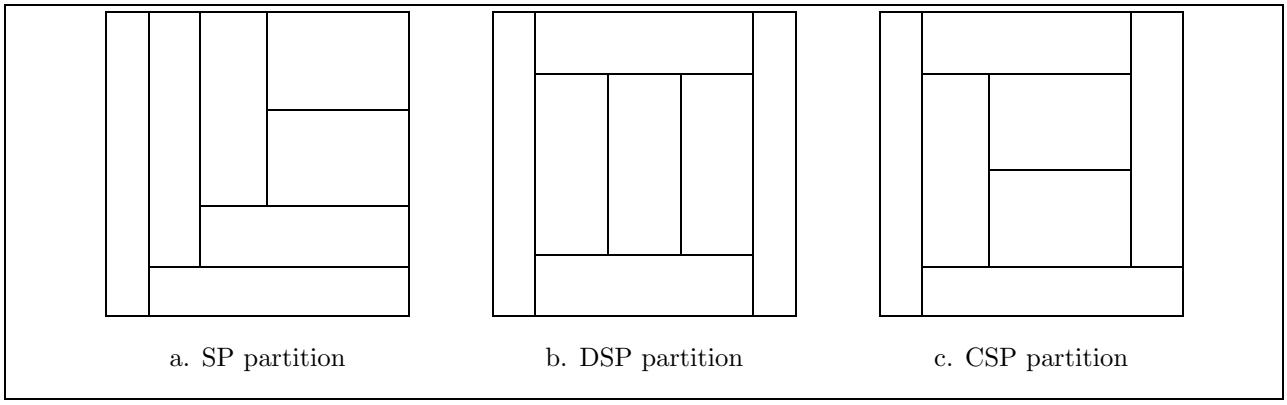


Figure 2: Unbalanced Partitioning Methods

In the next section, we investigate some unbalanced strategies which are promising strategies for high dimensional data with respect to their expected values.

3 Analysis of partitioning strategies for high dimensional data

In this section, we compare three unbalanced partitioning strategies SP(Sliced Partition), DSP(Dual Sliced Partition), CSP(Cyclic Sliced Partition). Two–dimensional case is shown in Figure 2. At each partition, all strategies partition d-dimensional data by a chosen one dimensional value and the other (d-1) dimensions are not considered as split axes. In the rest of this paper, we assume that N d-dimensional data are distributed in the unit hyper space $[0,1]^d$. Given a blocking factor *fanout*, we use the number of partitions P be $\lceil \frac{N}{fanout} \rceil$. Letters $\bar{\mathbf{q}}$ and q denote a query hyper cube and a query side length, respectively. In this section, we assume that data is uniformly distributed, thus each unbalanced partitioning strategy presented hereafter cyclically partitions data by one after another dimensions.

Partitioning strategies to be considered are as follow:

- SP(similar to TGS in (Garcia et al. 1997)) : For each dimension d_i, retrieve smallest $fanout$ data from a set D. Let an interval of the retrieved set of i^{th} dimension be [$low_{i,left}$,$high_{i,left}$]. Remember $high_{i,left}$. Among these distance values and associated dimensions, we select the dimension whose distance is the minimum and split a set D by a chosen dimension into two subsets D_1 and D_2 where $|D_1| = fanout$ and $D_1 = \{e_1 \mid e_1 \leq e_2, \forall e_2 \in D_2\}$. Generate a node encompassing a set D_1. Let D_2 be D and partition it recursively until we have P nodes.
- CSP : For each dimension d_i, retrieve smallest and largest $fanout$ data items from a set D. Let an interval of the retrieved set of smaller values be [$low_{i,left}$,$high_{i,left}$] and the retrieved set of larger values be [$low_{i,right}$,$high_{i,right}$]. Remember $high_{i,left}$ and $1-low_{i,right}$. Among these distance values and associated dimensions, we select the dimension whose distance is the minimum and split a set D by a chosen dimension and a direction into two subsets D_1 and D_2 where $|D_1| = fanout$ and $D_1 = \{e_1 \mid e_1 \leq e_2 \vee e_1 \geq e_2, \forall e_2 \in D_2\}$. Generate a node encompassing a set D_1. Let D_2 be D and partition it recursively until we have P nodes.
- DSP($Dual$–SP)(similar to (Berchtold et al. 1998)) :For each dimension d_i, retrieve smallest and largest $fanout$ data from a set D. Let an interval of the retrieved set of smaller values be [$low_{i,left}$,$high_{i,left}$] and the retrieved set of larger values be [$low_{i,right}$,$high_{i,right}$]. Remember $high_{i,left} + 1 - low_{i,right}$. Among these distance values and associated dimensions, we select the dimension whose distance is the minimum and split a set D by a chosen dimension into three subsets D_1, D_2, and D_3 where $|D_1| = fanout$, $D_1 = \{e_1 \mid e_1 \leq e_2, \forall e_2 \in D_2\}$, $|D_3| = fanout$, and $D_3 = \{e_3 \mid e_3 \geq e_2, \forall e_2 \in D_2\}$. Generate nodes encompassing two sets D_1 and D_3. Let D_2 be D and partition it recursively until we have P nodes.

In the following subsections, we show a brief example of each of them before we analyze. In our examples, we assume that $d = 5$ and $P = 10$. The i^{th} number in each parenthesis denotes the i^{th} dimensional extent of a MBH. The italicized number at i^{th} position denotes a partitioned or an assigned length at i^{th} dimension. Normal numbers denote unpartitioned lengths at its dimension. Note that the volume of each MBH generated is equal to $\frac{1}{P}$.

3.1 Analysis of SP

In this subsection, we introduce SP partitioning strategy by an example and present analytical results on it.

Example 3.1. *Since we assume uniform data distribution, SP cyclically partitions a data set from* 1^{st} *to* 5^{th} *dimension. The results are shown in Figure 3. Suppose* q = 0.7. *The expected numbers of* M_1 *is* $\frac{\frac{1}{10}}{1-\mathrm{q}}$, *since the length of all other dimensions is greater than* $1 - 0.7$. *Similarly, the expected value of* M_6 *is* $\frac{\frac{1}{10}+\frac{9}{50}}{1-\mathrm{q}} \times 1^4$. *Note that the expected value of* M_6 *is the sum of assigned lengths,* $\frac{1}{10}$ *and* $\frac{9}{50}$, *at* 1^{st} *dimension over* $(1 - \mathrm{q})$.

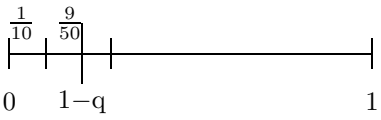
In the following lemmas, we present the assigned length of SP at each partition and present the expected value of it in terms of the assigned length.

Lemma 3.1. *Suppose that a d–dimensional MBH set M partitioned cyclically by SP is given. And let the number of partitions be* P_0 *and l be* P_0 **div** *d. Then the assigned length of* u^{th} *partition at* v^{th} *dimension* $P_{SP}(u, v)$ *is*

$$\begin{aligned} P_{SP}(1,v) &= \frac{1}{P_0 - v + 1} \\ P_{SP}(u,v) &= P_{SP}(u-1,v)\frac{P_{u-2} - v}{P_{u-1} - v + 1} \end{aligned} \quad (3)$$

M_1:$(\frac{1}{10}, 1, 1, 1, 1)$ M_2:$(\frac{9}{10}, \frac{1}{9}, 1, 1, 1)$
M_3:$(\frac{9}{10}, \frac{8}{9}, \frac{1}{8}, 1, 1)$ M_4:$(\frac{9}{10}, \frac{8}{9}, \frac{7}{8}, \frac{1}{7}, 1)$
M_5:$(\frac{9}{10}, \frac{8}{9}, \frac{7}{8}, \frac{6}{7}, \frac{1}{6})$ M_6:$(\frac{9}{50}, \frac{8}{9}, \frac{7}{8}, \frac{6}{7}, \frac{5}{6})$
M_7:$(\frac{36}{50}, \frac{2}{9}, \frac{7}{8}, \frac{6}{7}, \frac{5}{6})$ M_8:$(\frac{36}{50}, \frac{6}{9}, \frac{7}{24}, \frac{6}{7}, \frac{5}{6})$
M_9:$(\frac{36}{50}, \frac{6}{9}, \frac{14}{24}, \frac{3}{7}, \frac{5}{6})$ M_{10}:$(\frac{36}{50}, \frac{6}{9}, \frac{14}{24}, \frac{3}{7}, \frac{5}{6})$

a. Results of SP Partition

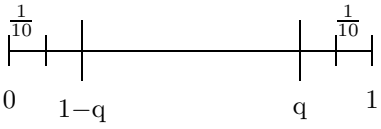


b. Assigned lengths at 1^{st} dimension

Figure 3: Example of SP partition when $P = 10$ and $d = 5$

M_1:$(\frac{1}{10}, 1, 1, 1, 1)$ M_2:$(\frac{1}{10}, 1, 1, 1, 1)$
M_3:$(\frac{8}{10}, \frac{1}{8}, 1, 1, 1)$ M_4:$(\frac{8}{10}, \frac{1}{8}, 1, 1, 1)$
M_5:$(\frac{8}{10}, \frac{6}{8}, \frac{1}{6}, 1, 1)$ M_6:$(\frac{8}{10}, \frac{6}{8}, \frac{1}{6}, 1, 1)$
M_7:$(\frac{8}{10}, \frac{6}{8}, \frac{4}{6}, \frac{1}{4}, 1)$ M_8:$(\frac{8}{10}, \frac{6}{8}, \frac{4}{6}, \frac{1}{4}, 1)$
M_9:$(\frac{8}{10}, \frac{6}{8}, \frac{4}{6}, \frac{2}{4}, \frac{1}{2})$ M_{10}:$(\frac{8}{10}, \frac{6}{8}, \frac{4}{6}, \frac{2}{4}, \frac{1}{2})$

a. Results of DSP Partition



b. Assigned lengths at 1^{st} dimension

Figure 4: Examples of DSP partition when $P = 10$ and $d = 5$

for $2 \le u \le l$, $1 \le v \le d$
where $P_i = P_0 - id$ *for* $1 \le i \le l$

Proof 3.1. *Let k be the total number of partitions at (u,v) and thus be* $(u-1)d+v$. *At each partition, the volume of data space is decreased by* $\frac{1}{P_0}$. *Thus after* $(k-1)$ *partitions, the volume is* $1 - \frac{k-1}{P_0}$. *Let* w_i *be the unpartitioned length at* i^{th} *dimension after* $(k-1)$ *partitions. Then*

$$\prod_{i=1}^{d} w_i = 1 - \frac{k-1}{P_0} \qquad \text{(a)}$$

Let L_v *be the assigned length at* v^{th} *dimension at* k^{th} *partition. Since the volume of MBH is equal to* $\frac{1}{P}$, *we have*

$$w_1 \dots L_v w_{v+1} \dots w_d = \frac{1}{P_0} \qquad \text{(b)}$$

By equating (a) and (b), we have

$$\frac{\prod_{i=1}^{d} w_i}{P_0 - k + 1} = w_1 \dots L_v w_{v+1} \dots w_d \qquad \text{(c)}$$

$$L_v = \frac{w_v}{P_0 - k + 1} \qquad \text{(d)}$$

Let $w_v^{'}$ *be the unpartitioned length at* $(u-2)^{th}$ *partition and* k_1 *be* $(u-2)d+v$. *Then the assigned length at* k^{th} *partition is*

$$L_v = w_v^{'} \frac{P_0 - k_1}{P_0 - k_1 + 1} \frac{1}{P_0 - k + 1} \qquad \text{(e)}$$

Substitute k_1 *and k, then we have*

$$L_v = P_{SP}(u-1, v) \frac{P_{u-2} - v}{P_{u-1} - v + 1} \qquad \text{(f)}$$

□

Now we present the expected value of SP.

Lemma 3.2. *Suppose that a d–dimensional MBH set M partitioned by SP and a* $\bar{\mathbf{q}}$ *with its side length q are given. Let u and v be the largest integers satisfying* $\sum_{i=1}^{u} \sum_{j=1}^{v} P_{SP}(i,j) \le 1 - \mathrm{q}$ *when* $\mathrm{q} \ge 0.5$. *Then the expected value* $E_{SP}(M_d, \bar{\mathbf{q}})$ *of SP when* $\mathrm{q} \ge 0.5$ *is*

$$\begin{aligned} E_{SP}(M, \bar{\mathbf{q}}) &= \frac{\sum_{i=1}^{u-1} \sum_{j=1}^{d} P_{SP}(i,j)}{1 - \mathrm{q}} \\ &+ \frac{\sum_{j=1}^{v} P_{SP}(u,j)}{1 - \mathrm{q}} \\ &+ (P - k) \end{aligned} \qquad (4)$$

where $k = (u-1)d + v$ *and* $P_0 = |M|$

Proof 3.2. *Instead of formal proof, we briefly show a sketch of proof. By lemma 3.1,* $P_{SP}(u,v)$ *is a monotone increasing function. Since the cumulative sum of lengths at* $(u,v)^{th}$ *partition is the maximum and it is less than* $1 - \mathrm{q}$ *among all* $u\prime \le u$ *and* $v\prime \le v$, *the sum of assigned lengths of all other dimensions is also less than* $1 - \mathrm{q}$. *Thus their intervals are contained in intervals* [0, 1-q] *or* [q, 1] *and their expected values are 1 by proposition 2.1. Since SP partitions cyclically only at one dimension, the expected value of MBHs until* $(u,v)^{th}$ *partition is the sum of one–dimensional assigned lengths. Since the intervals of assigned lengths after* $(u,v)^{th}$ *partition intersect* $[1 - \mathrm{q}, \mathrm{q}]$, *their expected values are 1 by proposition 2.1.* □

Note that the expected value of SP is expressed as the cumulative sum of one dimensional assigned lengths and the number of MBHs left.

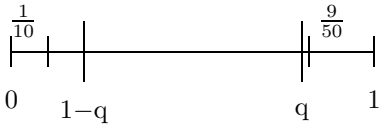
3.2 Analysis of DSP

Equation (4) denotes that the expected value is expressed as the sum of one-dimensional lengths and the number of MBHs. This fact leads us to develop a better heuristic called DSP, which avoids fast increase of accumulated lengths by partitioning both ends of a dimension. Before we show an analysis of DSP, we present an example to explain the DSP partitioning method .

Example 3.2. *At each partition, DSP cyclically splits both ends of a chosen dimension and thus it generates two MBHs(see Figure 4) at once. Two MBHs,* M_1 *and* M_2, *have the same assigned lengths* $\frac{1}{10}$ *at* 1^{st} *dimension. Their matching MBHs in example 3.1 are* M_1 *and* M_6. *The expected values of*

M_1:$(\frac{1}{10}, 1, 1, 1, 1)$ M_2:$(\frac{9}{10}, \frac{1}{9}, 1, 1, 1)$
M_3:$(\frac{9}{10}, \frac{8}{9}, \frac{1}{8}, 1, 1)$ M_4:$(\frac{9}{10}, \frac{8}{9}, \frac{7}{8}, \frac{1}{7}, 1)$
M_5:$(\frac{9}{10}, \frac{8}{9}, \frac{7}{8}, \frac{6}{7}, \frac{1}{6})$ M_6:$(\frac{9}{50}, \frac{8}{9}, \frac{7}{8}, \frac{6}{7}, \frac{5}{6})$
M_7:$(\frac{36}{50}, \frac{2}{9}, \frac{7}{8}, \frac{6}{7}, \frac{5}{6})$ M_8:$(\frac{36}{50}, \frac{6}{9}, \frac{7}{24}, \frac{6}{7}, \frac{5}{6})$
M_9:$(\frac{36}{50}, \frac{6}{9}, \frac{14}{24}, \frac{3}{7}, \frac{5}{6})$ M_{10}:$(\frac{36}{50}, \frac{6}{9}, \frac{14}{24}, \frac{3}{7}, \frac{5}{6})$

a. Results of CSP Partition



b. Assigned lengths at 1^{st} dimension

Figure 5: Example of CSP partition when $P = 10$ and $d = 5$

$E_{DSP}(1,1)$ and $E_{SP}(2,1)$ are 0.667($=\frac{0.1+0.1}{0.3}$) and 1.267($=\frac{0.1+0.1+0.18}{0.3}$), respectively when $\mathrm{q} = 0.7$. *Note that $E_{DSP}(1,1) < E_{SP}(2,1)$.*

Similar to SP, we present equations for the assigned length and the expected value of DSP in the following lemmas.

Lemma 3.3. *Suppose that a d-dimensional MBH set M partitioned by DSP is given. And let the number of partitions be P_0 and l be P_0* **div** *2d. Then the assigned length of u^{th} partition at v^{th} dimension $P_{DSP}(u,v)$ is*

$$P_{DSP}(1,v) = \frac{1}{P_0 - 2(v-1)}$$
$$P_{DSP}(u,v) = P_{DSP}(u-1,v)\frac{P_{u-2} - 2v}{P_{u-1} - 2(v-1)} \quad (5)$$

for $2 \le u \le l$, $1 \le v \le d$
where $P_i = P_0 - 2id$ for $1 \le i \le l$

Proof 3.3. *Similar to the proof of lemma 3.1.* □

Now we present the expected value of DSP.

Lemma 3.4. *Suppose that a d-dimensional MBH set M partitioned by DSP and a $\bar{\mathbf{q}}$ with its side length* q *are given. Let u and v be the largest integers satisfying $\sum_{i=1}^{u}\sum_{j=1}^{v} P_{DSP}(i,j) \le 1 - \mathrm{q}$ when $\mathrm{q} \ge 0.5$. Then the expected value $E_{DSP}(M,\bar{\mathbf{q}})$ of DSP when $\mathrm{q} \ge 0.5$ is:*

$$E_{DSP}(M,\bar{\mathbf{q}}) = 2\frac{\sum_{i=1}^{u-1}\sum_{j=1}^{d} P_{DSP}(i,j)}{1-\mathrm{q}} + 2\frac{\sum_{j=1}^{v} P_{DSP}(u,j)}{1-\mathrm{q}} + (P-k) \quad (6)$$

where $k = 2((u-1)d + v)$ and $P_0 = |M|$

Proof 3.4. *Similar to the proof of lemma 3.2.* □

Intuitively, the cumulative lengths of DSP is about half of that of SP method. Thus, $E_{DSP}(u,v)$ is approximately half of $E_{SP}(u,v)$. By analysis of DSP, we show again that the expected value is highly related to the cumulative sum of the one dimensional assigned lengths and the number of MBHs intersecting $[1-\mathrm{q}, \mathrm{q}]$.

3.3 Analysis of CSP

Analogous to previous analysis, we first show an example of CSP partition and present the expected value of it.

Example 3.3. *The results of cyclic partitions of CSP are shown in Figure 5 when $P = 10$ and $d = 5$. M_1 and M_6 have the same assigned lengths at 1^{st} dimension as in example 3.1. The sum of expected value of these is 0.933 ($= \frac{0.1+0.18}{0.3}$) when* $\mathrm{q} = 0.7$.

The assigned length of CSP is the same to that of SP since it partitions cyclically at each dimension. But, it partitions one of both ends of a dimension alternatively, the expected value is not the same. Now we present the expected value of CSP.

Lemma 3.5. *Suppose that a d-dimensional MBH set M partitioned by CSP and a $\bar{\mathbf{q}}$ with its side length* $\mathrm{q} \ge 0.5$ *are given. Let m_1, m_2 and v be the largest integers satisfying $\sum_{i=1}^{m_1}\sum_{j=1}^{v} P_{SP}(2i-1,j) \le 1-\mathrm{q}$ or $\sum_{i=1}^{m_2}\sum_{j=1}^{v} P_{SP}(2i,j) \le 1-\mathrm{q}$. Then the expected value $E_{CSP}(M,\bar{\mathbf{q}})$ of CSP when* $\mathrm{q} \ge 0.5$ *is*

$$E_{CSP}(M,\bar{\mathbf{q}}) = \frac{\sum_{i=1}^{m_1-1}\sum_{j=1}^{d} P_{SP}(2i-1,j)}{1-\mathrm{q}} + \frac{\sum_{j=1}^{v} P_{SP}(m_1,j)}{1-\mathrm{q}} + \frac{\sum_{i=1}^{m_2-1}\sum_{j=1}^{d} P_{SP}(2i,j)}{1-\mathrm{q}} + \frac{\sum_{j=1}^{v} P_{SP}(m_2,j)}{1-\mathrm{q}} + (P-k) \quad (7)$$

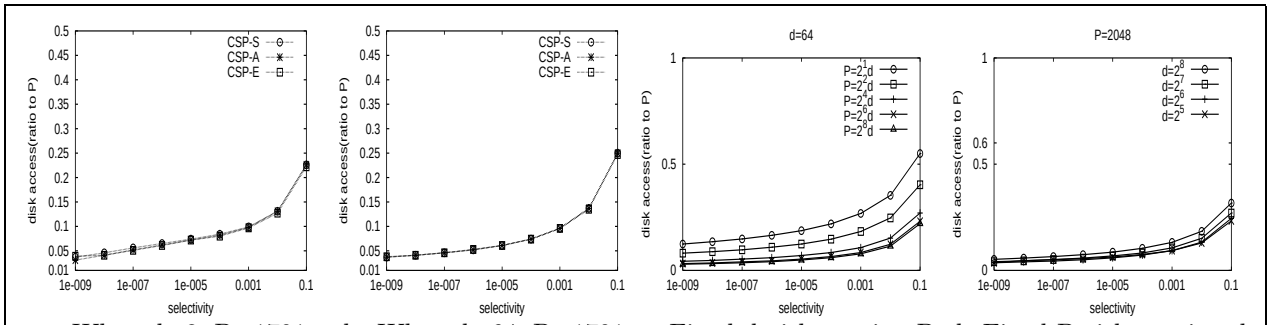
where $k = (u-1)d + v$ and $P = |M|$

Proof 3.5. *Similar to the proof of lemma 3.2.* □

The expected value of CSP is the cumulative sum of the assigned lengths from the left and the right end of a dimension. Similar to the expected value of SP and DSP, it is also expressed as the sum of assigned lengths and the number of MBHs intersecting the interval $[1-\mathrm{q}, \mathrm{q}]$ which is located around middle area of the data space.

3.4 Observations and Validation of Our Analysis

The expected value of SP, CSP, and DSP consists of the addition of two terms when $\mathrm{q} \ge 0.5$, that is $E_{method}(M,\bar{\mathbf{q}}) = \frac{\sum P_{method}(i,j)}{1-\mathrm{q}} + (P-k)$. When $\mathrm{q} < 0.5$, we approximate the expected value of them such as $E_{method}(M,\bar{\mathbf{q}}) = \frac{\sum P_{method}(i,j)}{1-\mathrm{q}}(1 + \frac{P-k}{P})$. Intuitively, the number and lengths of partitioned intervals of DSP and CSP in [0, q],[q,1-q], and [1-q,1] are almost the same and thus both strategies show similar behaviors and performances in simulations, analysis, and experiments. Thus we only consider the CSP method hereafter. To validate our analysis on CSP, we compare the expected value of it with generated MBHs which tile $[0,1]^d$ data space and with randomly generated data sets. Our comparisons are done with wide ranges of dimensions and of selectivities varying from 10^{-1} to 10^{-9} where N=68040 and fanout=40. Letters -S, -A, -E denote simulation, analysis, and experiment, respectively. Figures 6.a



a. When d=8, P=1701 b. When d=64, P=1701 c. Fixed d with varying P d. Fixed P with varying d

Figure 6: Analytical results and Experimental validation

and 6.b show that our analysis is remarkably identical to simulations and experiments. Both results show that the difference of the expected values is not significant even if dimension is increased by 8 times. The expected value is less sensitive to the increase of dimensionality, which is shown in Figure 6.d. What is sensitive to the expected value is the number of partitions at each dimension. But even if the number of partitions is multiplied by four time, the number of disk accesses is not increased by four. Its increment is less than four. In Figure 6.c, we show this fact.

As the summary of this section, we can conclude that the query performance of CSP is slightly influenced by dimensionality of the data space and is sublinearly proportional to the increase of the number of partitions.

4 Partitioning method and Associated data structure for high dimensional data

In the previous section, we pointed out that the query performance of DSP and CSP is almost identical when data is uniformly distributed. Since most of high dimensional data are skewed, we show some observations when the data distribution is skewed. Our observations show that CSP is better than DSP and thus we propose our new partitioning method based on CSP. We also observe that a MBH could be approximated by the order, the direction, and the split value of each partition.

4.1 Observations on skewed data distribution

In this subsection, we compare performance of DSP and CSP when the data distribution is skewed. The following example shows differences of DSP and CSP when the data distribution is mildly skewed.

Example 4.1. *Suppose the data distribution of a certain dimension is skewed as in Figure 7 and the number of partitions at this dimension is four. For simplicity, we assume that there are fanout data items in each interval C_i. And each length of C_1, C_2, and C_3 is a, each length of C_4 and C_5 is 2a, and 0.5 > 1-q > 4a. After four partitions, DSP generates four MBHs whose assigned lengths on this dimension are $\{C_1,C_5,C_2,C_4\}$. Thus the expected value of DSP is $(a+2a+2a+4a)/(1-\mathrm{q}) = 9a/(1-\mathrm{q})$. The result of CSP is $\{C_1,C_2,C_5,C_3\}$. The expected value of CSP is $(a+2a+2a+3a)/(1-\mathrm{q}) = 8a/(1-\mathrm{q})$. Thus, CSP is slightly better than DSP when data are mildly skewed on one dimension and are uniformly distributed on other dimensions.*

From an above example, we observe that when the data distribution is mildly skewed as in an example 4.1, CSP performs slightly better than DSP.

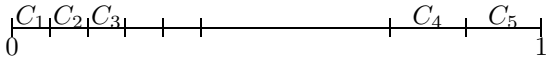


Figure 7: Mildly skewed data distribution

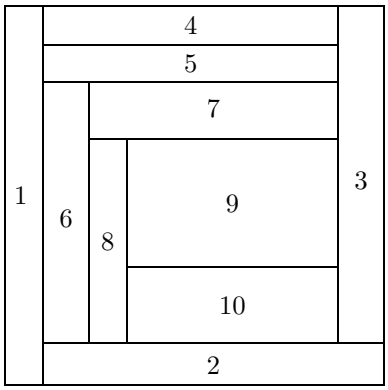


Figure 8: CSP partition

4.2 Observations on resulting MBHs

CSP partitions data from left-end or right-end to the center of data space. At each partition, the data space is divided into two mutually exclusive sets while the boundaries of other dimensions are determined by previous partitions. Figure 8 shows two–dimensional example of CSP. Each number in partitioned MBHs denotes the order of partitions, *op*. Assuming that *op* and a split value are given, then MBH can be approximated. For example, the 8^{th} MBH can be approximated by MBHs whose *op* is smaller than 8^{th} MBH.

Lemma 4.1. *Suppose a pair (k, $v_{i,\mathrm{dir},k}$) is assigned at each partition, it is sufficient to approximate MBH of k^{th} partition. A integer value k denotes op and a floating value $v_{i,dir,k}$ denotes the split value at i^{th} dimension when its split direction is* dir *at k^{th} partition.*

Proof 4.1. *Suppose that the* dir *is LEFT, then a value $v_{i,left,k}$ is the largest among all $v_{i,left,k_{prev}}$ where $k_{prev} < k$. Thus, the i^{th} dimensional extent of a MBH generated at k^{th} partition is $[v_{i,left,k_{prev}}, v_{i,left,k}]$ where k_{prev} is the largest integer satisfying $k_{prev} < k$. The intervals of other dimensions are $[0,1] - [0,v_{j,left,k_1}] - [v_{j,right,k_2}, 1]$ where $i \neq j$ and k_1, k_2 are the largest integers satisfying $k_1, k_2 < k$. The MBH of a pair $(k, v_{i,right,k})$ could be approximated similarly.* □

By the lemma 4.1, we show that a pair (k, $v_{i,dir,k}$) is sufficient to represent a MBH, which is an entry of

Dimension = 0

op	dir	split value
1	LEFT	0.1
3	RIGHT	0.88
6	LEFT	0.22
8	LEFT	0.32

Dimension = 1

op	dir	split value
2	LEFT	0.11
4	RIGHT	0.90
5	RIGHT	0.80
7	RIGHT	0.65
9	RIGHT	0.33
10	RIGHT	0.11

Figure 9: DSR of Figure 8

```
1.  Algorithm Distance-based Distribution-adaptive CSP
2.  input: input data D, fanout
3.  Initialize DSR;
4.  P ← ⌈|D| / fanout⌉, d_i ← 0, op ← 1;
5.  while(P > 0)
6.    for d_i ← d_i++ %d
7.      Sort D by d_i;
8.      left_dist ← GetDistance(d_i, LEFT, fanout, D);
9.      right_dist ← GetDistance(d_i, RIGHT, fanout, D);
10.     side ← LEFT, distance ← left_dist if left_distance < right_distance
11.     side ← RIGHT, distance ← right_dist, otherwise;
12.   partition_size ← fanout, done ← FALSE, chunksize ← 1;
13.   while(P>0 ∧ ∼done ∧ chunksize ≤ MAX_CHUNK_SIZE)
14.     next_dist ← GetDistance(d_i, side, partition_size, D);
15.     if(next_dist > distance + delta)
16.       done ← TRUE; continue;
17.     partition_size ← partition_size + fanout, P--, chunksize++;
18.     Cummulated_length[d_i].side ← Cummulated_length[d_i].side + val_{d_i,partition_size};
19.   CHUNK_FILE ← PartitionFile(d_i, side, partition_size, D);
20.   D ← D - CHUNK_FILE;
21.   InsertEntryIntoDSR(CHUNK_FILE, d_i, op++, side, DSR);
```

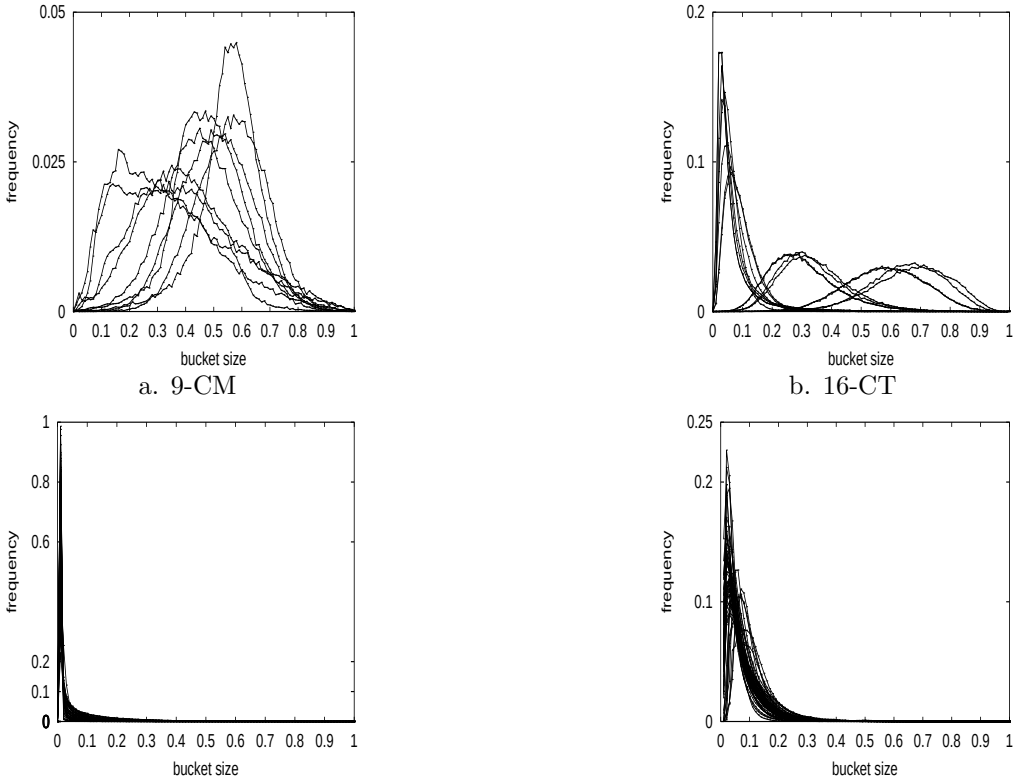
Figure 10: Algorithm DD–CSP

the DSR(Dimension-independent Single value Representation). DSR is a collection of these pairs. To help understanding of DSR structure, DSR of Figure 8 is shown in Figure 9. By the lemma, the 0^{th}–dimensional extent of the 8^{th} MBH is determined by an entry whose *op* is 6, that is [0.22, 0.32]. The 1^{st}–dimensional extent is determined by entries whose *op*s are 2 and 7.

Most of the tree–like index structures(White et al. 1996, Berchtold et al. 1996, Katayama et al. 1997, Böhm et al. 2001) use a (MBH, pointer) pair representation as an entry of a node. As a dimension grows, $fanout$ of these approaches is decreased by the factor of the dimension d with fixed page size or page size grows by the factor of d with fixed $fanout$. It is the common problems of the index structures which use dimension–dependent node representation like R-tree. Our representation is independent to dimension and is only dependent on the size of data. Assume that data size is 1,000,000, fanout is 50, and dimension is 30, the MBH representation requires about 4.8MB for terminal nodes whereas our approach requires 240KB. When dimension is 60, the MBH representation requires about 10MB whereas our approach requires the same amount of storage, 240KB. In the following subsection, we present our packing method, DD–CSP, as the conclusion of observations.

4.3 Proposed partitioning method and Associated data structure

In Figure 10, we show our algorithm in C–like pseudo code. A dimension is cyclically selected to partition a data set at line 7 instead of selecting a cost-efficient dimension among d dimensions. Since experimental studies show no significant performance gaps. An auxiliary function $val_{i,j}$ at line 18 returns a value of j^{th} element at i^{th} dimension. A function GetDistance(d_i,side, number, D) returns a value that $val_{d_i,number} - 0.0 + Cummulated\_length[d_i].left$ when side is LEFT or $1.0 - val_{d_i,|D|-number+1} + Cummulated\_length[d_i].right$, otherwise. Two system parameters MAX_CHUNK_SIZE and *delta* are introduces at lines 13 and 15, respectively. MAX_CHUNK_SIZE number of nodes are written to disk contiguously at maximum. A variable *delta* controls the level of clustering. We set MAX_CHUNK_SIZE = 10 and $delta = \max(\frac{1}{P}, 2(val_{d_i,partition\_size} - val_{d_i,partition\_size-fanout}))$. Thus, as data left are located around center area of the data space, MAX_CHUNK_SIZE number of nodes are stored contiguously. Note that it is performance-efficient to access MBHs *sequentially* which are located around center area by proposition 2.1. InsertEntryIntoDSR constructs an entry of DSR properly and stores a chunk of data, CHUNK_FILE,



a. 9-CM b. 16-CT

c. 32-CH d. 60-LANDSAT

Figure 11: Distribution of real data sets

contiguously in a secondary storage.

5 Experimental studies

In this section we compare DD–CSP to others, namely R-tree(Guttman 1984), TGS(Garcia et al. 1997), STR(Leutenegger et al. 1997), HP(Kamel et al. 1993), CSP, on both real and synthetic data sets. R-tree and TGS generate MBHs of all levels of nodes in top–down direction while DD–CSP, CSP, STR and HP partition the data sets in bottoms–up direction. CSP is a variant of DD–CSP whose *delta* = 0 and MAX_CHUNK_SIZE = 1. We only consider range queries on terminal nodes for comparison. Now we present data and query sets used in our experiments and show our results. Note that experiments on X–Tree and VA–FILE(Weber et al. 1998) are presented only in brief due to the lack of space.

5.1 Query and Data Sets

For query data sets, we select 1000 points from data sets and extend it by varying selectivities to generate hyper cubic queries. We varied the selectivities from 10^{-9} to 10^{-1}, corresponding to a side length of hyper cubic queries varying from $\sqrt[d]{10^{-9}}$ to $\sqrt[d]{10^{-1}}$. For the purpose of comparison, we fix *fanout* to be 40 irrelevant to dimensionality for all data sets except 60–LANDSAT data set. We fix *fanout* to be 60 for 60–LANDSAT.

We synthetically generate uniform data sets whose dimension varies from 8 to 64 whose size is 68040. Real data sets are 9-dimensional moments sets(9–CM), 16-dimensional texture sets(16–CT) and 32-dimensional color histogram sets(32–CH) which are extracted from the Corel images[1]. And we use 60-dimensional feature vectors(60–LANDSAT) of size 275465 representing texture information of blocks of large aerial photographs[2]. The distribution of these data sets are shown in Figure 11. It plots the frequency of the occurrences of the data in a given bucket size.

5.2 Performance Measure

Most researchers implicitly assume the cost of transferring a disk block negligible and thus only count the number of disk I/Os(positioning time) as a performance measure. With the fixed fanout, a node needs more disk blocks as dimension grows and thus the transferring cost is not negligible anymore. We incorporate the transferring cost into our disk access model as a ratio to positioning cost. A ratio α is defined as follows:

$$\alpha = \begin{cases} \frac{t_p}{fanout*d*2*sizeof(float)*t_x} & : \quad \alpha > 1 \\ 1 & : \quad otherwise \end{cases}$$

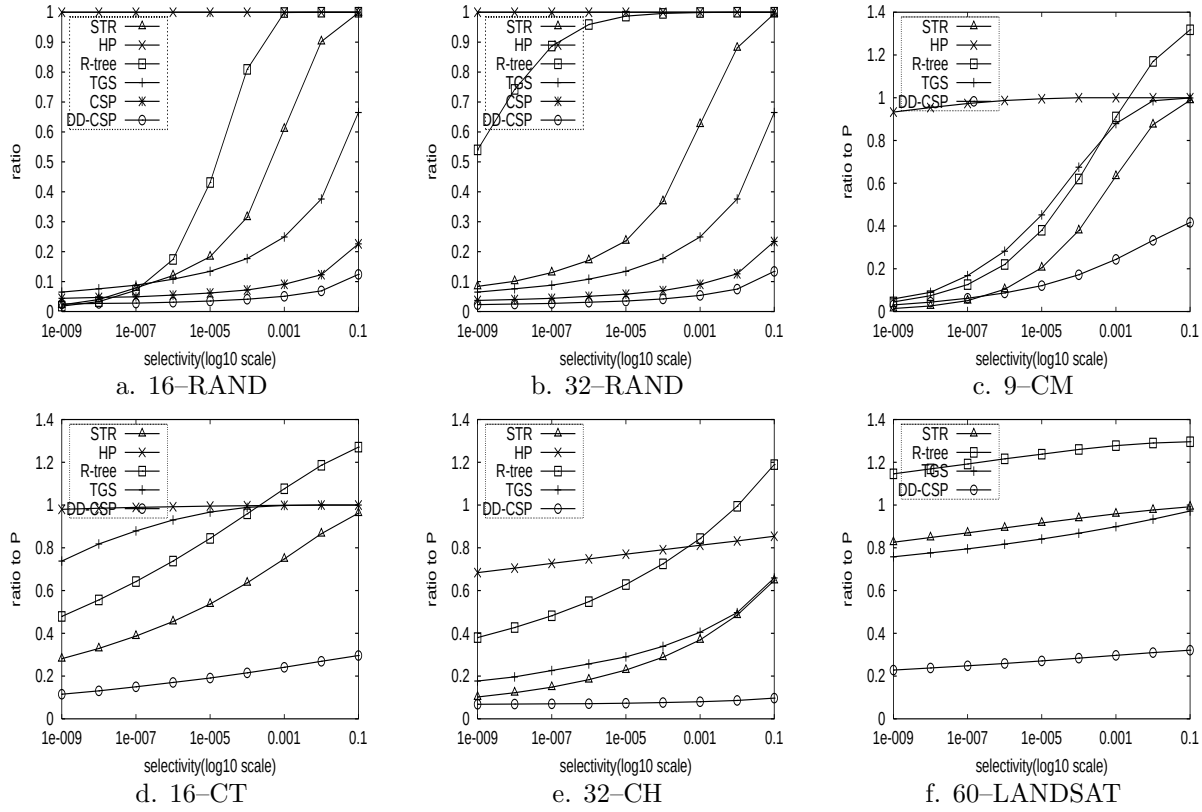
Parameters t_p and t_x denote positioning cost and transfer rate, respectively. A term α denotes the number of nodes can be transferred in a given positioning time. We use it to calculate the cost of transferring nodes contiguously stored in a secondary storage. The value of α is dependent on storage devices. In our experiments, we set $t_p = 1$ and $\alpha = 4$, for simplicity. Thus a single random access takes $(1+\frac{1}{\alpha})$ units of cost and k($\leq$MAX_CHUNK_SIZE) contiguous nodes accesses take $(1+\frac{k}{\alpha})$ units of cost. The total cost is expressed as a ratio to $P(1+\frac{1}{\alpha})$. The reader may refer (Ruemmler et al. 1994) for more details on secondary storage devices model.

5.3 Experiments

In this subsection we show the superior performance of our DD–CSP over R–tree, STR, TGS and HP. We

[1] kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.html

[2] vivaldi.ece.ucsb.edu/users/wei/gaborfeatures/airphotoFeatures



a. 16–RAND b. 32–RAND c. 9–CM

d. 16–CT e. 32–CH f. 60–LANDSAT

Figure 12: Experiments on real data sets

also show comparison to X–Tree and VA–FILE in short. We first experiment 16– and 32–dimensional synthetic datasets, namely 16–RAND and 32–RAND. The results of range queries on these sets(see Figure 12.a, .b) show that compared methods are competitive to DD–CSP for low dimensional data with smaller selectivities. But for high-dimensional data DD–CSP outperforms all other methods in all ranges of selectivities. Now we present experiments of all four real datasets, that is, 9–CM, 16–CT, 32–CH, and 60–LANDSAT. In all these experiments, DD–CSP uses $delta = \max(2\text{extent}, \frac{1}{P})$.

In all the experiments DD–CSP is a clearly winning strategy (see Figure 12.c–.f), achieving up to 136%(9–CM), 225%(16–CT), 567%(32–CH), and 202%(60–LANDSAT) savings in query response time over STR or TGS which are one of the best performing packing methods. The maximum gain is achieved for the 32–CH dataset. Since the data are highly condensed in a certain range and thus are packed(or clustered) up to MAX_CHUNK_SIZE. Thus we can conjecture that DD–CSP is advantageous for high dimensional data which are not only unform, but also skewed and condensed.

The method proposed in (Bercken et al. 1997) is not shown since the performance of (Bercken et al. 1997) is almost identical to R-tree(Bercken et al. 1997, Berchtold et al. 1998). Note that R–tree generates more terminal nodes than $P(=\frac{datasize}{fanout})$. We also show the part of experiments on high-dimensional indexes like X–Tree and VA–FILE. Table 1 shows the results for 32–CH data set. Numbers in the first row indicate selectivities. Measured CPU time is a ratio to linear scan.

DD–CSP is about four to eight times faster than X–Tree and about four to six times faster than VA–FILE. Experiments on other data sets also show the superiority of DD–CSP over other methods.

Table 1: Query processing time for 32-CH

	10^{-9}	10^{-7}	10^{-5}	10^{-3}	10^{-1}
VA–FILE	0.95	0.96	0.97	1.01	1.04
X–Tree	0.69	0.87	1.13	1.52	2.14
DD–CSP	0.16	0.16	0.17	0.19	0.25

6 Conclusion and Future Works

One of the most important decisions in packing is the selection of data partitioning strategies. For low-dimensional data, it is beneficial to partition data sets with equally sized MBRs. But, for high-dimensional data balanced partitioning strategies applicable to low-dimensional data are not beneficial anymore due to limited number of partitions. Thus, we focused on unbalanced partitioning strategies in our work. We analyzed several unbalanced strategies and presented a dominating metric in partitioning high dimensional data. We analytically showed that the one-dimensional cumulative length is directly related to the expected number of disk accesses. By simulations, we showed that our analysis was quite accurate under uniform data distribution assumption.

To adapt to data distribution, we generalized SP, CSP and DSP strategies and proposed a new partitioning method called DD-CSP. It selects a split dimension that incurs the least length increment and partitions (or clusters) data at one of both ends of a selected dimension to avoid fast increase of the cumulative length. We also presented a dimension–independent and light weighted data structure called DSR, which is highly coupled to DD–CSP. It takes constant amount of storage to represent the results of partitions, that is $c\frac{datasize}{fanout}$ where c $\approx$ 12Bytes. Its storage gains are a factor of d compared to traditional dimension dependent MBH–representations

which take about $8d\frac{datasize}{fanout}$.

We showed by experiments that DD–CSP is superior to all other strategies(that is, R-tree, HP, STR, TGS, X–Tree, and VA–FILE) in range queries with wide ranges of dimensions and of selectivities varying from 10^{-9} to 10^{-1}. And it is also very practical in a sense that it is highly storage efficient compared to other dimension dependent MBH–representations. Thus our method is a clearly winning packing method for high dimensional data in terms of range queries and storage requirements.

In summary, our work contributes in several aspects. First, we proposed cost models for several unbalanced partitioning methods and showed their accuracy of analysis by simulations and experiments on high-dimensional data space. Second, based on analysis, we suggested a simple partitioning method DD–CSP and showed by experiments its superiority over previously proposed packing methods on uniform and real data sets. Finally, we proposed a flat dimension-independent data structure DSR, which requires constant size of storage independent of dimension.

Future works could focus on extension of DD–CSP, providing the nearest neighbor search and declustering.

References

L. Arge. (1996), Efficient External-Memory Data Structures and Applications, Ph.d., *BRICS Dissertation Series, DS–96–03*, University of Aarhus.

L. Arge, K. Hindrichs, J. Vahrenhold & J.S. Vitter. (1999), 'Efficient Bulk Operations on Dynamic R-trees', *ALENEX*, pp. 328–348.

N. Beckmann, H.-P. Kriegel, R. Schneider & B. Seeger. (1990), 'The R*-tree: An Efficient and Robust Access Method for Points and Rectangles', *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pp. 322–331.

S. Berchtold, D.A. Keim & H.-P. Kreigel. (1996), 'The X-tree: An Index Structure for High-Dimensional Data', *Proc. 22rd VLDB Conf.* 28–39.

S. Berchtold, C. Bohm & H.-P. Kriegel. (1998), 'Improving the Query Performance of High-Dimensional Index Structures by Bulk Load Operations', *Proc. EDBT*.

J. van den Bercken, B. Seeger & P.W. Widmayer. (1997), 'A Generic Approach to Bulk Loading Multidimensional Index Structures', *Proc. 23rd VLDB Conf.* 406–415.

C. Böhm, S. Berchtold & D. Keim (2001), 'Searching in High-Dimensional Spaces-Index Structures for Improving the Performance of Multimedia Databases', *ACM Computing Surveys* **33**(3), pp. 322–373.

Y.J. Garcia, M.L. Lopez & S.T. Leutenegger. (1997), 'A Greedy Algorithm for Bulk Loading R-trees', *Technical Report 97–2*.

D.M. Gavrila. (1996), 'R-tree Index Optimazation', *Technical Report CS–TR–3292*.

A. Guttman. (1984), 'R-trees: A Dynamic Index Structure for Spatial Searching', *Proc ACM SIGMOD Int. Conf. on Management of Data* 47–57.

G.R. Hjaltason, H. Samet & Y. Sussmann. (1997), 'Speeding up bulk-loading of quadtrees', *ACM–GIS* 50–53.

G.R. Hjaltason & H. Samet. (1999), 'Improved Bulk-Loading Algorithms for Quadtrees', *ACM–GIS* 110–115.

I. Kamel & C. Faloutsos. (1993), 'On Packing R-trees', *Proc. Int. Conf. on Information and Knowledge Management(CIKM)* 490–499.

N. Katayama & S. Satoh. (1997), 'The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries', *Proc. ACM SIGMOD Int. Conf. on Management of Data* 369–380.

S.T. Leutenegger & D.M. Nicol. (1994), 'Efficient Bulk-Loading of Gridfiles', *ICASE Report 94–74*.

S.T. Leutenegger, M.A. Lopez & J. Edington. (1997), 'STR: A Simple and Efficient Algorithm for R-Tree Packing', *Proc. 13th Int. Conf. on Data Engineering(ICDE)*.

S.T. Leutenegger & M.A. Lopez. (1998), 'The Effect of Buffering on the Performance of R-Trees', *Proc. 14th Int. Conf. on Data Engineering(ICDE)* 164–171.

B.-U. Pagel, H.-W. Six, H. Toben & P.W. Widmayer. (1993), 'Towards an Analysis of Range Query Performance in Spatial Data Structures', *ACM PODS*.

B.-U. Pagel, H.-W. Six & M. Winter. (1995), 'Window Query-Optimal Clustering of Spatial Objects', *ACM PODS*, pp 86–94.

J.T. Roussopoulos & L. Leifker. (1985), 'Direct spatial search on pictorial databases using r-trees', *Proc ACM SIGMOD Int. Conf. on Management of Data*.

C. Ruemmler & J. Wilkes (1994), 'An Introduction to disk drive modeling', *IEEE Computer* **27**(3).

R. Weber, H.-J. & S. Blott. (1998), 'A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces', *Proc. 24th Int. Conf. on Very Large Databases*.

D. White & R. Jain. (1996), 'Similarity Indexing with the SS-tree', *Proc. 12th Int. Conf. on Data Engineering(ICDE)*.