QUILTS: Multidimensional Partitioning Framework Based on Query-Aware and Skew-Tolerant Space-Filling Curves

Shoji Nishimura
NEC Corporation
1588 Shimonumabe, Nakahara-ku, Kawasaki,
Kanagawa, Japan
s-nishimura@bk.jp.nec.com

Haruo Yokota
Tokyo Institute of Technology
2-12-1 Ookayama, Meguro-ku, Tokyo, Japan
yokota@cs.titech.ac.jp

ABSTRACT

Recently, massive data management plays an increasingly important role in data analytics because data access is a major bottleneck. Data skipping is a promising technique to reduce the number of data accesses. Data skipping partitions data into pages and accesses only pages that contain data to be retrieved by a query. Therefore, effective data partitioning is required to minimize the number of page accesses. However, it is an NP-hard problem to obtain optimal data partitioning given query pattern and data distribution.

We propose a framework that involves a multidimensional indexing technique based on a space-filling curve. A space-filling curve is a way to define which portion of data can be stored in the same page. Therefore, the problem can be interpreted as selecting a curve that distributes data to be accessed by a query to minimize the number of page accesses. To solve this problem, we analyzed how different space-filling curves affect the number of page accesses. We found that it is critical for a curve to fit a query pattern and be robust against any data distribution. We propose a cost model for measuring how well a space-filling curve fits a given query pattern and tolerates data skew. Also we propose a method for designing a query-aware and skew-tolerant curve for a given query pattern.

We prototyped our framework using the defined query-aware and skew-tolerant curve. We conducted experiments using a skew data set, and confirmed that our framework can reduce the number of page accesses by an order of magnitude for data warehousing (DWH) and geographic information systems (GIS) applications with real-world data.

1. INTRODUCTION

There has recently been a significant increase in the collection and analysis of large amounts of data, such as website logs and sensor data from various devices. Many data analysis tasks require only a small part of the collected data set. Therefore, *data skipping* [25, 31] is attracting increasing attention in solving data access bottlenecks. The key idea is to partition data into pages and access only pages containing data to be retrieved by a query. Since

SIGMOD'17, May 14 - 19, 2017, Chicago, Illinois, USA
© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4197-4/17/05. . . $15.00
DOI: http://dx.doi.org/10.1145/3035918.3035934

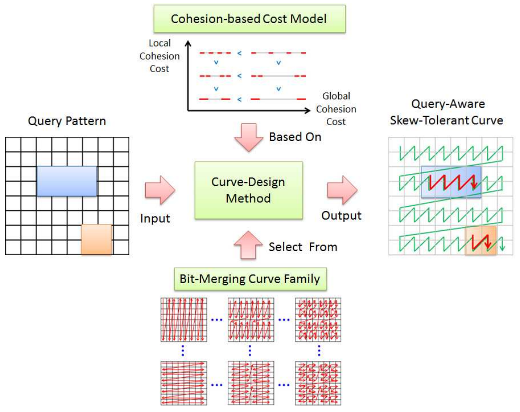


Figure 1: Overview of QUILTS, A Multidimensional Data Partitioning Framework based on a Query-Aware and Skew-Tolerant Space-Filling Curve

the query processing performance is proportional to the number of page accesses, data partitioning is critical to reduce page accesses under given query patterns and data distributions. Unfortunately, obtaining optimal data partitioning is an NP-hard problem [31].

Sun et al. [31] proposed a bottom-up approach based on a clustering algorithm to achieve near-optimal data partitioning. Their approach runs a tailored clustering algorithm that uses a query pattern and data distribution and associates each page with some metadata, such as the minimum and maximum of each attribute value in the page. Their approach significantly reduces the number of page accesses during a query execution due to their clustering algorithm. However, their applications are restricted to archived data because data partitioning is done as a pre-processing task and requires full metadata access to skip pages during query processing.

A top-down approach using a space-filling curve [16, 23] is another option. This approach transforms a multidimensional data point into a single value in accordance with the visiting order of a space-filling curve and uses a B+-Tree or range-partitioned key value store, such as HBase, as an underlying index. This approach can update data partitioning upon data update because of the underlying index and using an efficient query processing algorithm [16, 30]. However, this approach requires selecting an appropriate space-filling curve to minimize the number of page accesses. There are extensive studies on selecting the optimal space-filling curve for multidimensional data [19, 18, 17, 20, 35]. Most studies have targeted on ad-hoc queries, a general case, and do not pay much attention to optimizing the query processing performance for given query patterns or constraints. More importantly, to the best of our knowledge, the characteristics of space-filling curves on different data distributions are hardly discussed, although data distribution affects query processing performance significantly.

We analyzed such characteristics and found that it is critical for a curve to fit a query pattern and be robust against any data distribution. We say that a curve is *query-aware and skew-tolerant* when the curve holds the above properties for a given query pattern and can handle skew data distributions.

Thus, we propose a multidimensional data partitioning framework based on a query-aware and skew-tolerant space-filling curve, called **QUILTS**: **QU***ery*-**I***ntensive* **L***inearization* **T***olerating data* **S***kew*. The overview is illustrated in Figure 1. We prototyped the framework and conducted experiments using a skew data set. We then confirmed that QUILTS reduced the number of page accesses by an order of magnitude for applications with real-world data.

1.1 Contributions

The technical contributions of the paper are as follows:

Cohesion-based cost model. We propose a cost model to measure how well a space-filling curve fits a given query pattern and tolerates skew data distributions to minimize the number of page accesses. We identify the characteristics of the C-curve, also known as the composite index, and the Z-curve based on the proposed cost model. We found that the two curves have opposite characteristics on a given query pattern and data distribution.

Bit-merging curve family. We define a bit-merging curve family, which is a generalization of the C- and Z-curves. Although the two curves have opposite characteristics, according to their definitions, their visiting orders are both defined as merging bit-sequences of attribute values. We thus synthesize them into a curve that inherits both their characteristics by changing the order of bit-merging.

Curve-design method. We propose a heuristic method to design a query-aware and skew-tolerant curve from the bit-merging curve family. The size of the bit-merging curve family exhibits combinatorial explosion. Therefore, a brute force approach does not work for optimal curve selection. We derive a heuristic based on cost measurements of various bit-merging curves.

1.2 Organization

The rest of the paper is organized as follows. Section 2 reviews related literature. Section 3 introduces the foundation of space-filling-curve-based data partitioning. Section 4 describes our cost model to identify the characteristics of space-filling curves. Section 5 describes our method of designing a query-aware and skew-tolerant curve for a given query pattern. Section 6 describes experiments and comparisons. Future work and conclusions are presented in Sections 7 and 8.

2. RELATED WORK

Data Skipping. Data skipping [25, 31] partitions data into pages and associates each page with metadata, such as the minimum and maximum values in the page. When processing a query, it skips pages irrelevant to the query. Since the performance gain depends on the number of skipped pages, Sun et al. [31] proposed a data-partitioning method based on a query-aware clustering algorithm. It effectively partitions data into pages to minimize the number of page accesses, but the application is restricted to archived data because data partitioning is performed as a pre-processing task.

Space-Filling Curve based Indexing. A space-filling curve [27, 4] is a common technique to implement a multidimensional index combined with a one-dimensional index, such as the B+-index [28, 16, 13, 34, 6, 5], or a range-partitioned key value store such as HBase [36, 23, 11].

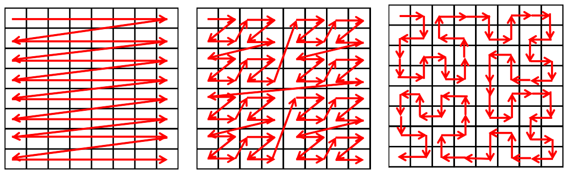

(a) C-curve (b) Z-order (c) Hilbert curve

Figure 2: Popular Space-Filling Curves

The C-curve is a conventional multidimensional index widely used in relational databases. Since the composite index concatenates multiple attribute values to a single value, it is also called a C-curve from the viewpoint of a space-filling curve, as shown in Figure 2(a). Query awareness is critical for the composite index because the concatenation order affects query performance [32]. The selectivity of the first attribute generally affects performance the most. Thus, the composite index works well when the selectivity of each attribute is heterogeneous. However, in case of homogeneous selectivity, it is difficult to achieve the best performance.

The Z-curve, also known as the Morton order [22] (Figure 2(b)), and Hilbert curves [10] (Figure 2(c)) are popular space-filling curves for spatial indexes because these curves can handle the homogeneous selectivity of each attribute well. It is assumed that all dimensions have the same domain size. However, this assumption is inconsistent with real-world applications. Zero-padding is a well-known technique to fit domain size. However, it causes unnecessary space consumption. Markl [15] proposed a variation of the Z-curve that eliminates the tailing zero-padding and introduced the idea of surrogate keys to shrink the key length. Hamilton et. al. [7] extended the Hilbert curve to eliminate the heading zero-padding. These techniques take into account the query constraint, but they focus on the reduction of space cost, not that of access cost.

Analysis on Space-Filling Curves Several space-filling curves have been used for data management besides those mentioned above. There have been several studies on analyzing the characteristics of space-filling curves to determine proper curves for effective data management. Mokbel et al. [19] defined a set of preferable characteristics that the space-filling curves should have, and derived a closed form of each characteristic in the general case. Then, they analyzed the characteristics based on the dimensionality. Finally, they proposed *irregularity* that evaluates *locality-preservation*, i.e., how close a pair of neighboring points in a multidimensional space will become when mapped to a space-filling curve [17]. Moon et al. [20] proposed the *clustering number* that models the number of disk seeks during range query processing. A multidimensional query is interpreted as a set of one-dimensional queries on a space-filling curve. Therefore, the size of the set represents the number of disk seeks. Pan et al. [35] recently proved that the Hilbert curve is one of the best curves with respect to the clustering number in the general case. These analyses are focused on selecting the best curve in a general case, not in a specific case such as for a given query pattern. Moreover, these analyses are not focused on the relationship between a space-filling curve and the data distribution.

3. BACKGROUND

3.1 Multidimensional Data and Query

Multidimensional data are defined as objects with multiple attributes and are represented as points in a multidimensional space (Figure 3). For example, location data are a kind of multidimen-

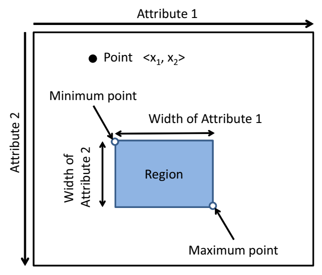


Figure 3: Multidimensional Space

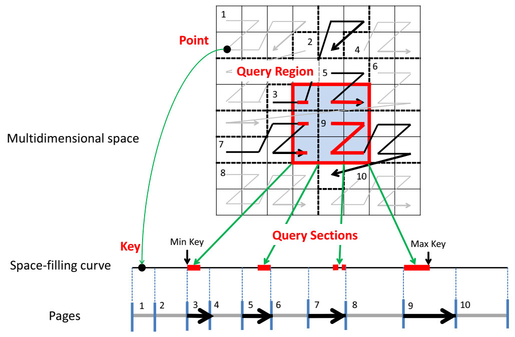


Figure 4: Mapping Between Multidimensional Space and Space-Filling Curve

sional data, and in a two-dimensional space, their attributes are longitude and latitude. Each dimension refers to the corresponding domain of each attribute. Formally, for a data point with d attributes, we represent it as $\langle x_1, \ldots, x_d \rangle$, where x_i is the value of attribute i.

A multidimensional range query retrieves data satisfying a set of range conditions with respect to (w.r.t.) its attributes. Each range condition is described as $[x_i^s, x_i^e]$ where x_i^s (resp. x_i^e) is the start (resp. end) value of the range condition w.r.t. attribute i. We assume that the start and end values are inclusive for ease of discussion. We represent the range conditions of a multidimensional range query using a *query region* in the multidimensional space (Figure 3). Then, we define a *query pattern* based on the shape of the region, i.e., the width of the region in each dimension. Specifically, a query pattern is a collection of queries with the same range widths in all dimensions. In other words, we define a query pattern using a combination of range condition widths. For example, assuming W_i is the range condition width of attribute i, the query pattern is denoted as $W_1 \times W_2 \times \ldots \times W_d$. We call the point constituted by start (resp. end) values in all dimensions $\langle x_1^s, x_2^s, \ldots, x_d^s \rangle$ (resp. $\langle x_1^e, x_2^e, \ldots, x_d^e \rangle$) *the minimum* (resp. *maximum*) *point* of a region and represent the region using the pair of its minimum and maximum points.

Answering multidimensional range queries is challenging in that we cannot compare two multidimensional points then define a unique order over the multidimensional space. In a one-dimensional space, we can sort all points by their attribute values then process a range query efficiently. However, multidimensional points such as $\langle 1, 2 \rangle$ and $\langle 2, 1 \rangle$ are not comparable; thus, we cannot define a total order over these points. As a result, it is difficult to organize multidimensional data for efficient range query processing.

3.2 Space-Filling-Curve-Based Index

A space-filling curve is a one-dimensional representation of a multidimensional space. Figure 4 depicts the mapping between a multidimensional space and space-filling curve. Each point in the multidimensional space is mapped as a *key* on the space-filling curve. The order of a key is defined as the order that the point will be visited by this curve. A query region in the multidimensional space is transformed to a set of *query sections* on the space-filling curve. Therefore, a multidimensional range query is mapped as a set of range queries on the space-filling curve. The mapping depends on the type of space-filling curve applied. In other words, we can control the layout of query sections on a space-filling curve by applying a different type of space-filling curve.

The space-filling-curve-based index is a technique for implementing a multidimensional index on top of a one-dimensional index such as B+-Tree. Each multidimensional point is transformed to a key on a space-filling curve and is indexed by the key. The one-dimensional index partitions keys into pages by key range. In Figure 4, the space-filling curve is split into ten key ranges, and each key range is associated with a page. For example, region 1 in the multidimensional space in Figure 4 is associated with page 1. The length of a key range depends on the data distribution. If the data distribution is dense, the length will be small since the one-dimensional index maintains pages by fixing the number of keys in each page.

The straightforward range-query algorithm using the space-filling-curve-based index decomposes a query region in a multidimensional space into a set of query sections on a space-filling curve and obtains the final result by summarizing the processing result of each query section. However, some queries may generate a combinatorial number of small query sections incurring high computation cost. To solve this problem, several sophisticated algorithms [33, 26, 30, 13] process queries by *page* rather than query section. For example, the algorithm determines pages 3, 5, 7, and 9 based on the query region in Figure 4, and processes the query by scanning the 4 pages rather than the 5 query sections.

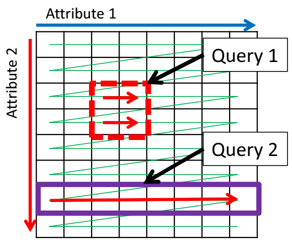
3.3 Curve Selection

The performance of query processing depends on the number of page accesses, which further depends on the query pattern and data distribution. In this section, we discuss how a query pattern and data distribution affect the number of page accesses and the importance of selecting proper space-filling curves to minimize the number of page accesses.

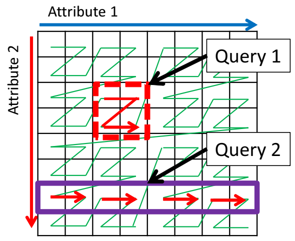
3.3.1 Query Pattern and Query-Aware Curve

Figure 5 shows how a query pattern can affect the number of page accesses. Queries 1 and 2 represent two extreme query patterns. Query 1 (the region with broken lines) has the same range widths in both dimensions, while the range widths of Query 2 (the region with solid lines) take either the minimum or maximum value. For Query 1, the Z-curve seems better compared to the C-curve because the trace of the Z-curve is continuous within the query region while that of the C-curve is split. In the case of Query 2, however, the C-curve seems better than the Z-curve for a similar reason. This observation indicates that each space-filling curve has its favorable query patterns.

The problem of selecting proper space-filling curves for arbitrary query patterns has been treated as curve selection for ad-hoc queries. A curve for an ad-hoc query should be robust enough to handle any kind of query pattern, even though it may not be optimized for the query. Moreover, in real-world applications, it is common to have explicit or implicit range constraints. For example, in business intelligence applications users can logically specify

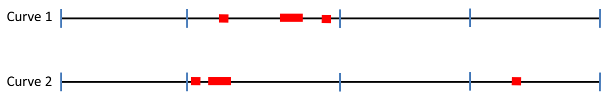


(a) C-curve

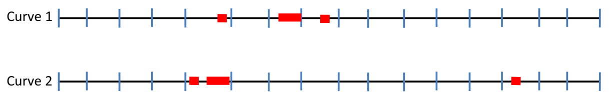


(b) Z-curve

Figure 5: Space-Filling Curves and Query Patterns



(a) Sparse Distribution Case



(b) Dense Distribution Case

Figure 6: Query Section Layout and Data Distributions

any scale of range width for the time dimension, but a practical upper limit will probably be less than 1 year. Such range constraints can be used to optimize query processing performance. We discuss the details in Section 5.

3.3.2 Data Distribution and Skew-Tolerant Curves

In this section, we discuss how data distribution affects the selection of space-filling curves.

First, data distribution affects the key range of each page. Each page has its capacity. When the number of keys exceeds the capacity, the page is split into two pages. The page that a key will be assigned to is determined by the middle key in the original page. If the key is before the middle key, it will be assigned to the first page; otherwise, the second page. In other words, each page stores the same number of keys. However, their key ranges may differ. If a page stores keys mapped from the dense region of a data distribution, its key range will be narrow. On the other hand, if a page covers the sparse region of a data distribution, its key range will be wide. Therefore, if a data distribution is skew, some pages will have narrow key ranges and some pages will have wide key ranges.

Next, the desirable mapping property of a space-filling curve may change depending on the data distribution, as shown in Figure 6. The thick lines represent query sections mapped onto Curves 1 and 2, and the short vertical lines represent page boundaries. In the case of a sparse data distribution, pages cover wider key ranges (Figure 6(a)). All query sections on Curve 1 are covered within a single page, while those on Curve 2 are two pages. Therefore, Curve 1 is expected to exhibit better query processing performance than Curve 2 because it accesses one less page. In the case of a dense data distribution, pages cover narrower key ranges (Figure 6(b)). Query sections on Curve 1 are covered by three pages, while those on Curve 2 are two pages. Therefore, Curve 2 is expected to perform better than Curve 1. Thus, if data density changes, the desirable mapping property will also change.

As discussed above, a skew-data distribution leads to key ranges of different lengths covered by each page, and it also affects the desirable mapping properties used to minimize the number of page accesses during query processing. However, an important observation is that these mapping properties can coexist with each other for limited types of query patterns. Our idea is to apply a space-filling curve holding desirable mapping properties for both the dense and sparse data distributions. The curve can reduce the number of page accesses during query processing against both a dense and sparse region. In other word, the curve can handle a skew data distribution. We thus call such a curve a *skew-tolerant* curve.

Page capacity also affects the key range of each page. If the page capacity is small, the key range tends to be narrow. Therefore, a skew-tolerant curve can also handle different page capacities.

Although the characteristics of space-filling curves have been extensively studied, to the best of our knowledge, there are no models to identify the characteristics of space-filling curves against data distribution. One challenge is how to design a model to identify these characteristics. Another challenge is how to find a skew-tolerant curve based on the model.

3.4 Challenges

We discussed that the selection of space-filling curves significantly affects query-processing performance. The above observations indicate that we should select a space-filling curve that fits a target query pattern and tolerates a skew-data distribution to minimize the number of page accesses. However, it is challenging to find a query-aware and skew-tolerant space-filling curve.

Throughout the following sections, we discuss our framework of multidimensional data partitioning using query-aware and skew-tolerant space-filling curves. First, we propose a cost model to measure how well a space-filling curve fits a given query pattern and tolerates data skewness in Section 4. Then, we discuss our method of designing a query-aware and skew-tolerant curve for a given query pattern in Section 5.

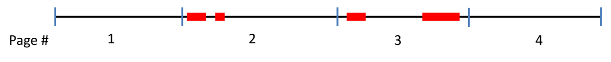
4. COHESION-BASED COST MODEL

This section describes designing a cost model for identifying the characteristics of space-filling curves. We observed that query-processing performance depends on the number of page accesses. First, we analyze which characteristics of space-filling curves affect the number of page accesses. Then, we investigate the relationship between the trajectory of a space-filling curve and query pattern as well as a data distribution. Finally, we propose a cost model to measure how well a space-filling curve fits a query pattern and tolerates a skew-data distribution.

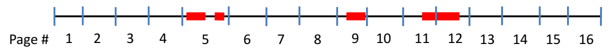
4.1 Analysis of Page Access Patterns

For page-based query processing algorithms, query-processing performance depends on the number of page accesses. This number depends on the data distribution because the coverage of each page changes according to the data distribution. We use the models shown in Figure 7 to analyze which characteristic of a curve affects the number of page accesses.

First, we analyze a sparse data distribution case (Figure 7(a)). Each page tends to cover a wide range due to the small number of page splits. Since each page covers a wide range, the query sections tend to be covered with continuous pages (Pages 2 and 3 in Figure 7(a)). If the intervals between query sections are shorter, we can expect a smaller number of page accesses. Therefore, a space-filling curve is preferable in the case of sparse-data distribution if the curve globally clusters query sections mapped from a query region.



(a) Sparse Data Distribution Case



(b) Dense Data Distribution Case

Figure 7: Data Distributions and Query Sections

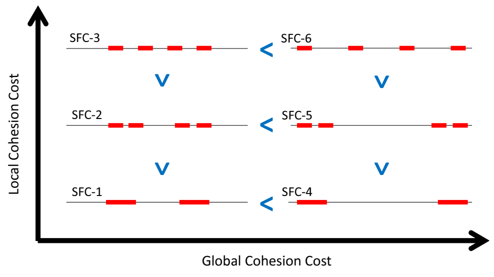


Figure 8: Cohesion Based Cost Model

Next, we analyze the case of a dense data distribution (Figure 7(b)). In this case, each page tends to cover a narrow range due to the large number of page splits. When each page covers a narrow range, the query sections tend to be covered with separate pages (Pages 5, 9, 11, and 12 in Figure 7(b)). Since page-based query processing algorithms are designed to scan only pages covering query sections, a long interval between query sections does not affect the number of page accesses. For example, the number of accessed pages does not change even if query sections in Page 5 in Figure 7(b) shift to Page 1. In other words, if query sections are clustered locally, we can expect a smaller number of page accesses. Therefore, a space-filling curve is preferable in the case of a dense data distribution if the curve locally clusters query sections mapped from a query region.

In summary, we should consider two types of clustering characteristics of a space-filling curve on different data distributions to minimize the number of page accesses. In a sparse data distribution case, it is preferable that a curve globally clusters query sections mapped from a query region. In a dense data distribution case, it is preferable that a curve locally clusters these query sections.

4.2 Cohesion-Based Cost Model

We propose a cost model based on the above analysis. The cost model is illustrated in Figure 8. The horizontal axis represents the cost based on the global clustering property of a space-filling curve. The vertical axis represents the cost based on the local clustering property of a space-filling curve. The right and upper directions mean higher costs. For example, space-filling curve 1 (SFC-1 in the figure) has a lower cost than SFC-4 w.r.t. the cost based on the global clustering property, and SFC-1 has a lower cost than SFC-2 and SFC-3 w.r.t. the cost based on the local clustering property. We define *global cohesion cost* to measure the global clustering property, and *local cohesion cost* to measure the local clustering property. The details are discussed in the following sections.

4.2.1 Global Cohesion Cost

Global cohesion measures how much the query sections are clustered globally. In other words, global cohesion evaluates the distance between the minimum and maximum keys of a query region (see Figure 4). The distance consists of the lengths of the query sections and those of the intervals between the query sections. Since the lengths of query sections are equivalent to the area of the query region, they are independent of the selected space-filling curve. However, the lengths of intervals depend on the selected curve. Thus, we define the global cohesion cost using the interval length:

DEFINITION 1 (GLOBAL COHESION COST). *Global cohesion cost C_G is defined as the sum of all interval lengths between query sections. Let n_i be the length of the i-th interval, such that*

$$C_G = \sum_i n_i. \tag{1}$$

Global cohesion cost is closely related to the locality-preserving property introduced in Section 2, but their views of closeness are different. The locality-preserving property measures how close two neighbor points in a multidimensional space are when mapped onto a space-filling curve. On the other hand, global cohesion measures how close points in a query region are when mapped onto a space-filling curve.

4.2.2 Local Cohesion Cost

Local cohesion measures how much the query sections are clustered locally. There are two aspects of the local clustering property: the number of query sections and variance of the interval lengths between query sections, as shown in Figure 8. For example, SFC-1 in the figure has a smaller number of query sections than SFC-2, and SFC-2 has a larger variance of the interval lengths than SFC-3. To measure both aspects, we define local cohesion as the entropy of the ratio of the interval lengths between query sections:

DEFINITION 2 (LOCAL COHESION COST). *Local cohesion cost C_L is defined as the entropy of the ratio of the interval lengths between query sections. Let n_i be the length of the i-th interval such that*

$$C_L = -\sum_i \frac{n_i}{\sum_i n_i} \log \frac{n_i}{\sum_i n_i}. \tag{2}$$

Note than C_L is 0 where no interval or a single interval exists.

Local cohesion cost is closely related to the clustering number introduced in Section 2. The clustering number is used to evaluate the number of query sections. On the other hand, local cohesion is used to more aggressively evaluate the closeness of adjacent query sections.

4.2.3 Total Cohesion Cost

We defined two cost measurements, global and local cohesion. These two measurements can be used to compare SFC-1 and SFC-4 in Figure 8 in accordance with global cohesion and to compare SFC-1 and SFC-2 or SFC-3 in accordance with local cohesion. When we want to compare SFC-2 and SFC-4, we need a mixed measurement of both global and local cohesion costs. Thus, we define *total cohesion cost* as follows:

DEFINITION 3 (TOTAL COHESION COST). *Total cohesion cost C_T is defined as a multiplication of C_G and C_L. Let n_i be the i-th interval between query sections, such that*

$$\begin{aligned} C_T &= C_G \cdot C_L \\ &= (\sum_i n_i)(-\sum_i \frac{n_i}{\sum_i n_i} \log \frac{n_i}{\sum_i n_i}) \\ &= (\sum_i n_i) \log(\sum_i n_i) - \sum_i (n_i \log n_i) \end{aligned} \tag{3}$$

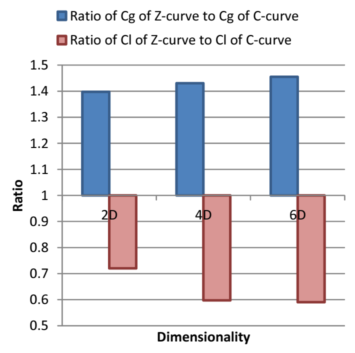


Figure 9: Cohesion Costs Comparison between C- and Z-curves

Strictly speaking, we should give priority to either the global or local cohesion cost in accordance with the data distribution. For example, either the global or local cohesion cost should be weighted if the data distribution is sparse or dense, respectively. The mixed model is simple, but it is sufficient to capture the overall property of a space-filling curve w.r.t. the number of page accesses. If we need to understand the detailed property of a space-filling curve against data distribution, we can decompose the total cohesion cost into global and local cohesion costs.

4.3 Cohesion Costs and Skew-Tolerance

We defined two cost models to capture the mapping property against data distribution. The global cohesion cost is used to evaluate the clustering property against a sparse data distribution and the local cohesion cost is used against a dense data distribution. If a curve can reduce both costs, it is expected to reduce the number of page accesses against any data distribution.

DEFINITION 4 (SKEW-TOLERANT CURVE). *A skew-tolerant curve is a curve that reduces both global and local cohesion costs.*

4.4 Cost Analysis of C- and Z-curves

We analyze C- and Z-curves using the cohesion-based cost model. We observe that these two curves show opposite characteristics for a specific query pattern and data distribution.

First, we evaluated the average cohesion costs of the C- and Z-curves for random query patterns. We generated 100 queries whose ranges were selected at random. The dimensionality ranged from 2 to 6. Each domain size was 6 bits (from 0 to 63). Figure 9 summarizes the comparison results of the average global and local cohesion costs between the C- and Z-curves. The bar represents the ratio of average cohesion cost of the Z-curve to that of the C-curve. When the ratio is greater than 1, it means that the C-curve is better than the Z-curve; otherwise, the Z-curve is better. We observed that these curves have opposite properties regarding cohesion costs. Figure 9 shows that the C-curve reduces the global cohesion cost by about 40% compared to the Z-curve; on the other hand, the Z-curve reduces the local cohesion cost by about 40% compared to the C-curve. This tendency can be observed even if the dimensionality increases. From these results, we can conclude that the C-curve is better in reducing the number of page accesses than the Z-curve in a sparse data distribution case, but the Z-curve is better in a dense data distribution case.

Then, we evaluated the cohesion costs of the C- and Z-curves for specific query patterns. We generated 100 queries for 7 query patterns: 2×128, 4×64, 8×32, 16×16, 32×8, 64×4, and 128×2. These query patterns have the same area size. The results are summarized in Figure 10. Both the global and local cohesion costs of the C-curve were the lowest in the case of the 2×128 query pattern, and were the highest in the case of the 128×2 query pattern. While the local cohesion cost of the Z-curve remained constant against all query patterns, the global cohesion cost was minimum in the case of the 16×16 query pattern and increased as the query pattern became further from a square. These results indicate that curves have preferred query patterns.

In summary, the C- and Z-curves have opposite results w.r.t. cohesion cost. The results indicate that the C-curve operates best for a sparse data set and elongated rectangle-like query patterns, while the Z-curve operates best for a dense data set and square-like query patterns.

5. CURVE DESIGN

This section introduces our method of designing skew-tolerant curves for a given query pattern. In the previous section, we analyzed the C- and Z-curves using the cohesion-based cost model and found that the C-curve tends to be skew-tolerant for elongated rectangle-like query patterns, and the Z-curve prefers square-like query patterns. If a query pattern is an intermediate of these query patterns, *what curve achieves the most skew tolerance?* In the rest of this section, first we define the bit-merging curve family, a generalization of the C- and Z-curves. Then, we analyze the relationship between query patterns and bit-merging curves. Finally, we propose our method of designing a skew-tolerant curve for a given query pattern.

5.1 Bit-Merging Curve Family: Generalization of C-curve and Z-curve

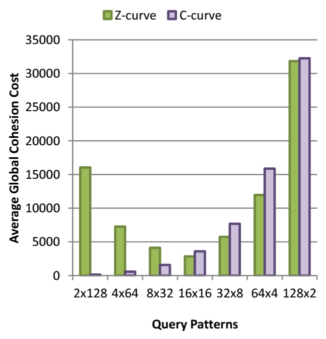
Now, we define *the bit-merging curve family*, a set of curves generalizing the C- and Z-curves. The C- and Z-curves seem to be completely different types of curves. However, the construction of the two curves is quite similar. Both curves are constructed by merging bit-sequences in all dimensions: the C-curve concatenates bit-sequences, while the Z-curve interleaves bit-sequences. For example, let a two-dimensional point be a pair of bit-sequences $\langle \mathtt{x_1x_2x_3x_4}, \mathtt{y_1y_2y_3y_4} \rangle$. The C-curve defines the order as $\mathtt{x_1x_2x_3x_4}$ $\mathtt{y_1y_2y_3y_4}$, and the Z-curve defines as $\mathtt{x_1y_1x_2y_2x_3y_3x_4y_4}$. The difference is how to merge bit-sequences. Thus, we generalize the bit-merging order and define the *bit-merging curve family* as follows.

DEFINITION 5 (BIT-MERGING CURVE FAMILY). *A bit-merging curve is a curve that defines multidimensional points by merging bit-sequences in all dimensions. A bit-merging curve family is the set of all possible bit-merging curves.*

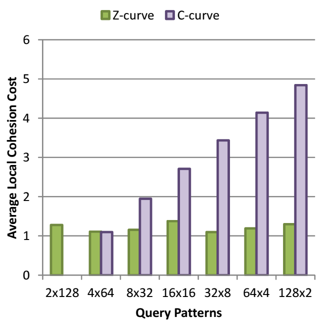
Figure 11 shows examples of the bit-merging curve family in a two-dimensional space. The horizontal axis represents a spectrum of bit-merging order from bit-concatenation to bit-interleaving and the vertical axis represents the precedence of attributes. The bit-merging curve family contains a combinatorial explosive number of curves. Let n_i be the bit-length of attribute i; then the size of the curve family is represented as the multinomial coefficient [8][1]

$$\frac{(\sum_{0 \le i < d} n_i)!}{\prod_{0 \le i < d} (n_i!)}.$$

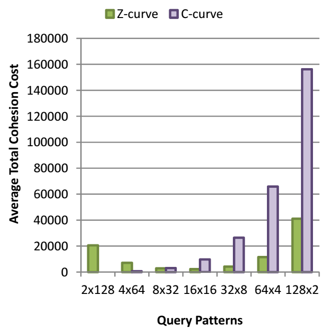
[1]This problem is equivalent to counting the number of ways of locating $n(= \sum n_i)$ different elements (the total bit-length of attributes) in d different cell (the number of attributes) in which cell i contains n_i elements (the bit-length of each attribute) without taking the order of the elements in any cell into account [8].



(a) Global Cohesion Cost



(b) Local Cohesion Cost



(c) Total Cohesion Cost

Figure 10: Cohesion Cost Analysis of Query Patterns with Constant Area Size

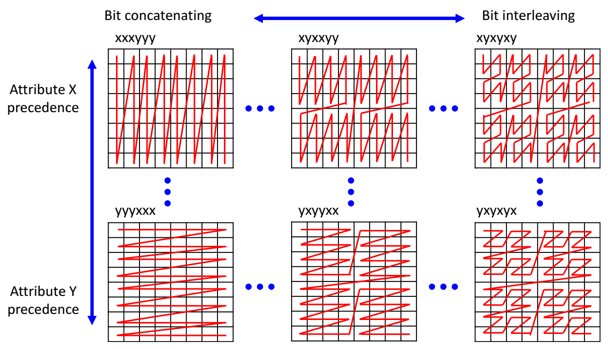


Figure 11: Bit-Merging Curve Family

For example, when the target space is two-dimensional and each domain size is 4 bits, the size is $70(=\frac{8!}{4!\cdot 4!})$. If the domain size increases from 4 to 8 bits, then the size grows to $12870(=\frac{16!}{8!\cdot 8!})$. If the dimensionality increases from 2 to 4, then the size explodes to $63063000(=\frac{16!}{4!\cdot 4!\cdot 4!\cdot 4!})$.

5.1.1 Curve Notation

A curve in the bit-merging curve family can be identified by the merging order of bit-sequences. Thus, we name each curve in the curve family after the bit-merging order like *curve* $\mathtt{x_1y_1y_2y_3y_4x_2x_3x_4}$. Since the curve name tends to be long, we use a short notation to save space. We omit each suffix and apply a run-length compression. For example, curve $\mathtt{x_1y_1y_2y_3y_4x_2x_3x_4}$ is denoted as curve x1y4x3 because the bit-merging order is one bit comes from attribute x, four bits come from attribute y, and finally three bits come from attribute x.

We can use parentheses to group a bit-sequence. For example, the Z-curve $\mathtt{x_1y_1x_2y_2x_3y_3x_4y_4}$ is denoted as curve (xy)4, repeating bit-sequence xy four times.

5.2 Query Pattern and Bit-Merging Order

The bit-merging curve family expands a design space explosively. However, it is not practical to evaluate the cohesion costs of all the bit-merging curves for a given query pattern. We analyze what type of bit-merging curve tends to reduce cohesion costs for a given query pattern.

We first analyze why the C- and Z-curves are skew-tolerant for elongated rectangle-like and square-like query patterns, respectively. Figure 12 shows the trajectories of the C- and Z-curves and six query patterns. From the cohesion based cost analysis, we find that the C-curve prefers a query pattern, as shown in Figure 12(a), and the Z-curve prefers a query pattern, as shown in Figure 12(e). When we observe the curve trajectories and query patterns, we find that trajectories in Figures 12(a) and 12(e) did not break within the query region. If the trajectory of a curve does not break within a query region, we say that the curve holds *trajectory continuity* within the query region.

We continue to determine what condition is required to achieve trajectory continuity within the query region. We analyze the relationship between the query range width in each dimension and the bit-merging order. The width of 8×2 in bit length is 3 bits for the horizontal dimension and 1 bit for the vertical dimension because 8 is 2^3 and 2 is 2^1. On the other hand, the C-curve in Figure 12(a) is represented as curve yyyxxx, where x and y are for the horizontal and vertical dimensions, respectively. Since the width in bit length is 3 bits $\times$ 1 bit, we determine where three xs and one y are located in yyyxxx. Then, we determine that three xs and one y occupy the least significant four bits. This is the condition of trajectory continuity within the query region.

To validate the condition, we analyze the cases in Figures 12(b) and 12(e). In both cases, the width in bit length is 2 bits $\times$ 2 bits. We determine where two xs and two ys are located in yyyxxx and yxyxyx. We then find that these four bits do not occupy the least significant four bits of yyyxxx, but occupy that of yxyxyx. This is why the trajectory within the region in Figure 12(b) breaks and that in Figure 12(e) continues.

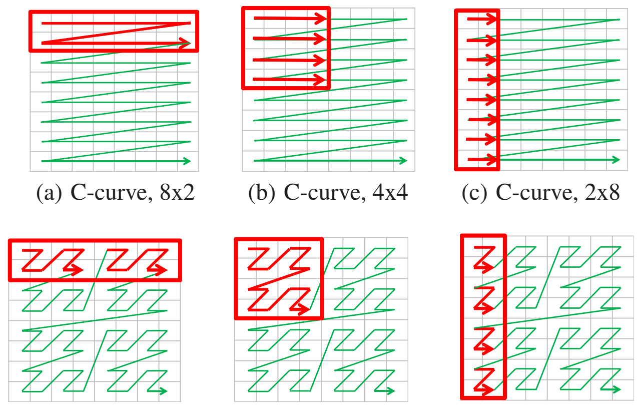
Next, we analyze whether a curve will be skew-tolerant if it satisfies the trajectory-continuity condition. We generate 80 bit-merging curves and measure their total cohesion costs. Figure 13 summarizes the top 25 curves that reduce the total cohesion cost for query pattern 8×32. If a curve satisfies the condition, the bar of the curve is shown as solid. Figure 13 shows that curves satisfying the condition are top-ranked. Therefore, we conclude that we should select curves satisfying the trajectory-continuity condition.

5.3 Curve-Design Method

We propose a curve design method based on the relationship between the query pattern and the bit-merging order. First, we discuss the case of a single query pattern. Then, we extend the method to multiple query patterns. Finally, we discuss a-priori information needed for curve design.

5.3.1 Curves for Single Query Pattern

Our proposed method is used to design a skew-tolerant curve for a given single query pattern. We discussed that a curve becomes skew-tolerant if the curve satisfies the trajectory-continuity condi-


(a) C-curve, 8x2 (b) C-curve, 4x4 (c) C-curve, 2x8

(d) Z-curve, 8x2 (e) Z-curve, 4x4 (f) Z-curve, 2x8

Figure 12: Query Patterns and Trajectory Continuity

tion within the query region in Section 5.2. We use the trajectory-continuity condition to design a curve for a single query pattern. The algorithm is illustrated in Figure 14 and is described as follows.

1. Calculate the bit length of the query-range width in each dimension.

2. Round up the bit length of the query-range width if it is fractional.

3. Let d_i be the bit length of the query-range width in the i-th dimension.

4. In each dimension, split bits into the least significant d_i bits and remaining bits. The former goes to the least significant group and the later goes to the most significant group.

5. Merge bits in the least and most significant parts of all dimensions to form the new bit-merging curve.

At the last step of the algorithm, any bit-merging order is applicable. However, we recommend intermediate bit-merging between the C- and Z-curves because such a bit-merging curve will moderately inherit properties from both the C- and Z-curves.

5.3.2 *Curves for Multiple Query Patterns*

We extend the curve design method to handle multiple query patterns. First, we observe hierarchical structures of a bit-merging curve then combine the observation and trajectory-continuity condition to extend the method for multiple query patterns.

Figure 15 shows the hierarchical structures of curve `xxyyyx`. The figure indicates that curve `xxyyyx` can be a skew-tolerant curve for query patterns 2×4, 2×8, 4×8, and so on. Each bit-merging curve has its own set of query patterns in which the curve can be skew-tolerant. Therefore, if we control the bit-merging order, we can determine the set of query patterns.

We use the hierarchical structures and trajectory-continuity condition to extend our curve design method for multiple query patterns. Figure 16 illustrates this method, and the algorithm is described as follows.

1. Calculate the bit lengths of the largest and smallest query-range widths in each dimension.

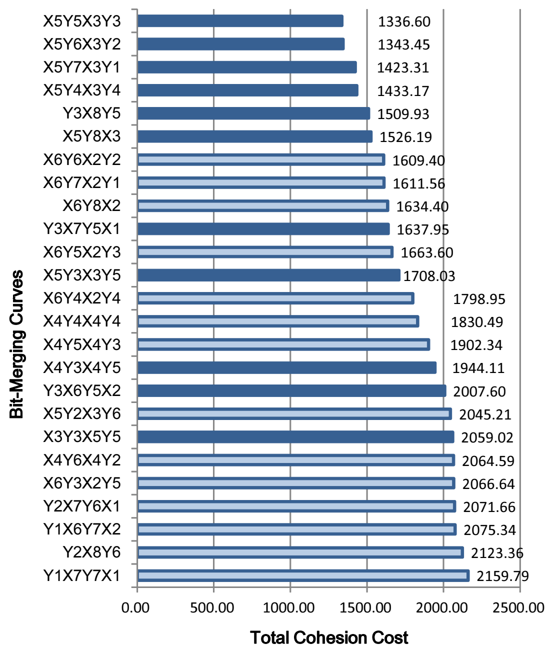


Figure 13: Top 25 Candidate Curves for 8 × 32 Queries

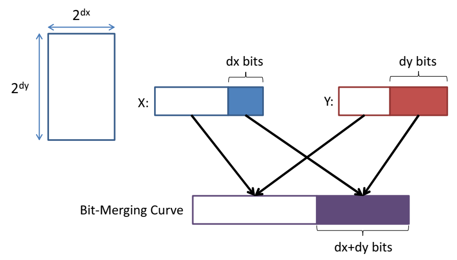


Figure 14: Curve for Single Query Pattern

2. Round up the bit length of query range width if it is fractional.

3. Let u_i and l_i be the bit lengths of the largest and smallest query-range widths in the i-th dimension, respectively.

4. In each dimension, split bits into three parts, the most significant, intermediate, and least significant in accordance with u_i and l_i.

5. Merge bits in each part of all dimensions.

6. If there is an intermediate query-range width between the largest and the smallest query-range widths, apply the same steps for the intermediate part.

When a curve is designed for multiple query patterns, we can prioritize a frequent query pattern over others. With this extended method, the bit-merging order within each part does not matter. Therefore, we can prioritize a frequent query pattern freely. The idea is to combine it with the original method. We apply it for the frequent query pattern and obtain d_i. Note that $l_i \leq d_i \leq u_i$. Then, we determine a bit-merging order satisfying the condition of l_i, d_i, and u_i. Thus, we can not only fit a curve to multiple query patterns but also prioritize a frequent query pattern.

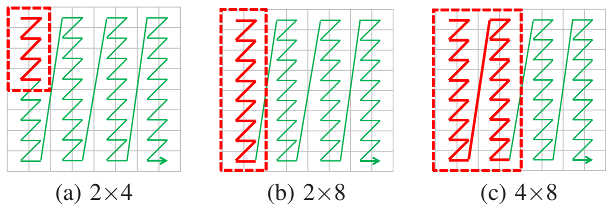


Figure 15: Hierarchical Structures of Curve xxyyyx

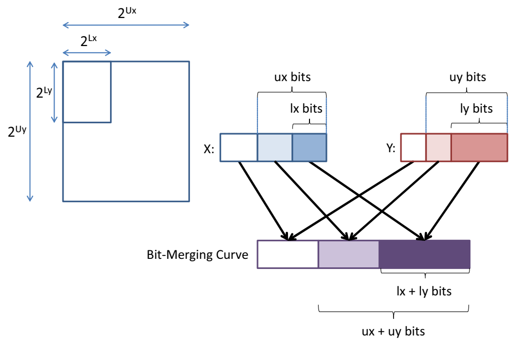


Figure 16: Curve for Multiple Query Patterns

Note the following limitation of the curve design method. The method can design a skew-tolerant curve for the largest and the smallest query patterns, but it gives up some intermediate query patterns. For example, we should give up either $Lx \times Uy$ or $Ux \times Ly$ in Figure 17.

5.3.3 A-priori Information for Curve Design

The curve design method requires a-priori information about query patterns. The most important information is the typical query range width in each dimension. However, the method does not require the exact query range width. It is sufficient to know the width in the range from half to double the real width because the query range width is rounded up to the nearest power of two at design time.

Although this may be a limitation of the proposed method, it is feasible to obtain such information in many applications.

One example is a data warehouse (DWH) application. This application summarizes collected data by aggregation unit such as day, week, and month in the time dimension, and by category in the product dimension. The aggregation unit is typical information about a query pattern. We can define any aggregation unit logically, but we choose specific ones in practice. Note that the proposed method requires information about range width, not the start or end point of the range. Therefore, the proposed method can be applied to not only a case where start and end points are statically fixed such as from 0:00 to 23:59 of any day, but also in which those points are dynamically specified such as the last 24 hours.

Another example is a geographical information system (GIS) application. We can logically specify any spatial range from worldwide (the maximum width) to the accuracy limit of a GPS (the minimum width). In practice, however, applications have their own limit such as from citywide to one block. These upper and lower bounds of range width are typical representations of a query pattern.

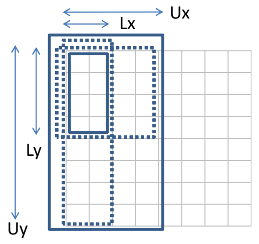


Figure 17: Limitation of Curve-Design Method for Multiple Query Patterns

Of course, some applications can take any query range width, and the width might change over time. For the first case above, we recommend choosing the Z-curve to minimize the variance of the total cohesion cost against any query pattern. If we can expect the magnitude of change, such as from 1/2, 1/4, or 1/8 to 2, 4, or 8, we can design a curve in accordance with this expectation for the second case.

6. EVALUATION

We conducted experiments to evaluate the proposed cost model and the curve selection heuristics. The first experiment involved evaluating how well the proposed cost model captures the properties of the curve against data distribution. The second experiment involved evaluating how much a curve designed with our curve method reduces the number of page accesses.

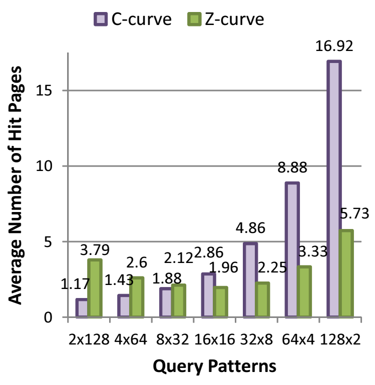
We implemented QUILTS on top of HBase, a range-partition-based distributed key-value store [9]. We extended the algorithm for the Z-curve designed by Skopal et al. [30] to handle the bit-merging curve family. We omit the details of the algorithm due to space limitation.

6.1 Evaluation of Cohesion Based Cost Model

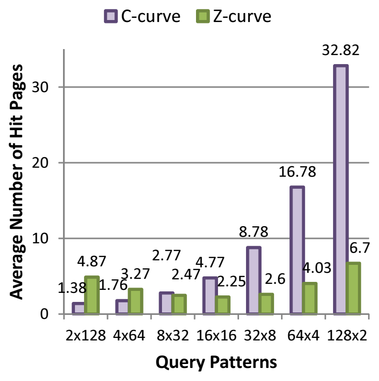
We evaluated how well the proposed cost model captures the curve properties against query patterns and data distribution. We observed the number of page accesses against the data distribution. We synthesized uniform distributed data sets to minimize the effect of query location and control data distribution by changing the size of the data set from 10 million (M) records as a sparse data distribution case to 40 M records as a dense data distribution case.

We selected the C- and Z-curves for evaluation because the two curves have opposite properties against the proposed cost model. We selected seven query patterns as shown in Figure 10. We randomly generated 100 queries for each query pattern. Since we selected uniform distributed data sets and query patterns with the same query region size, we expected that each query would retrieve a similar number of hit records from the same data set, and that the number of hit records would be proportional to the size of the data set. Therefore, the number of page accesses against a query pattern shows how neatly a curve fits the query pattern and the increase in the number of page accesses against data size indicates the robustness or sensitivity of the curve against data distribution. Figure 18 summarizes the execution results of seven query patterns using three data sets. The horizontal axis represents the type of query patterns, and the vertical axis represents the average number of page accesses per query.

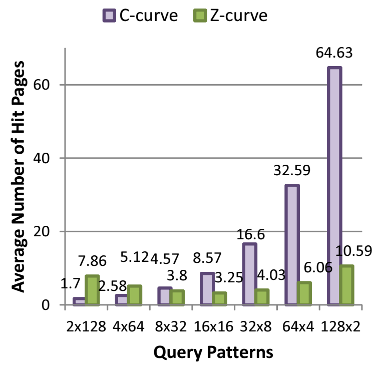
The number of page accesses was closely related to the total cohesion cost. The C-curve minimized the number of page accesses in the 2×128 query pattern and maximized the number in the 128×2 query pattern. In both Z- and C-curves, we could observe



(a) Data Size: 10 M



(b) Data Size: 20 M



(c) Data Size: 40 M

Figure 18: Number of Page Accesses per Query Pattern

the relationship between the number of page accesses and total cohesion cost. The Z-curve minimized the number of accessed pages in the 16×16 query pattern, and the number of page accesses increased as the query pattern went to 2×128 or 128×2. In addition, as we can see from the total cohesion costs of the Z-curve, the number of page accesses w.r.t. the 128×2 query pattern was larger than that of the 2×128 query pattern. By comparing the results of the C- and Z-curves, we could find which curve fits more neatly to a query pattern from the total cohesion cost. For example, the total cohesion cost showed that the C-curve is appropriate for the 2×128 and 4×64 query patterns, while the Z-curve is appropriate for the 16×16, 32×8, and 128×2 query patterns, and both curves are competitive for the 8×32 query pattern. As expected, the number of page accesses is related to the total cohesion cost.

Then, we compared the C-curve, Z-curve, and Curve `x3y8x5` based on the average number of page accesses w.r.t. the 8×32 query pattern. Curve `x3y8x5` is a curve between the C- and Z-curves, and is skew-tolerant for the 8×32 query pattern ranked in Figure 13. We changed the data distribution by changing the data set size from 10 to 40 M. Figure 19 summarizes the results. The results of Curve `x3y8x5` are similar to or better than those of the C- and Z-curves. Therefore, Curve `x3y8x5` is more robust than the C- and Z-curves w.r.t. data distribution.

The C- and Z-curves had similar total cohesion costs against the 8×32 query pattern. However, the global and local cohesion costs of the two curves were completely different. The Z-curve showed a lower local cohesion cost but a higher global cohesion cost compared to the C-curve. Figure 19 shows that the C-curve reduced the number of page accesses compared to the Z-curve when the data size was 10 M. However, when the data size increased to 20 and 40 M, the Z-curve performed better. This indicates that the proposed cohesion cost model can accurately capture the properties of a space-filling curve on data distribution.

6.2 DWH Application

We conducted experiments to evaluate the efficiency of our curve design method through retail sales analysis, a typical DWH application.

We defined a data model based on a case study introduced in a previous study [12]. Table 1 summarizes the details of the data set. In accordance with this study, the data model in retail sales has three dimensions: date, product, and store. The product and store dimensions have hierarchical structures. The product dimension consists of layers of department, category, and brand, and the

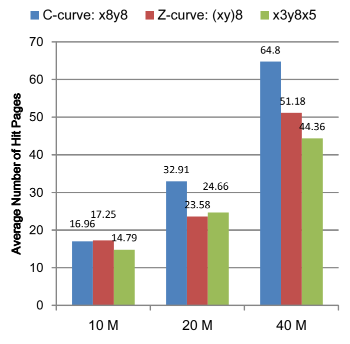


Figure 19: Number of Page Accesses w.r.t. 8×32

Table 1: Data Model for Retail Sales Analysis

Dimension	Layered Attribute	Bit Length	Distribution
Date		32	Poisson
Product		(32)	
	Department	8	Zipf
	Category	10	Zipf
	Brand	14	Zipf
Store		(32)	
	Region	6	Normal
	Store	10	Normal

store dimension consists of layers of region and individual stores. We assigned an appropriate bit length for each layer in accordance with its cardinality and configured an appropriate data distribution. Note that the data set contains various levels of data distribution because the product dimension is highly skewed due to the Zipf distribution, and the store dimension is moderately skewed due to the Normal distribution.

The typical workload is to roll up to an aggregation unit such as department and region. We assume that the date dimension rolls up to day, week, and month, the product dimension to department and category, and the store dimension to region. Thus, we define six query patterns by combining the aggregation units in each dimension. Table 2 summarizes the bit lengths of the query-range width.

We defined four curves. The first curve is curve `s16d32p32`, where `s` means the store dimension, `d` the date dimension, and `p`

Table 2: Aggregation Unit and Query-Range Width

Date	Product	Store
day (16.4 bits)	category (14 bits)	region (10 bits)
week (19.2 bits)	department (24 bits)	
month (21.3 bits)		

the product dimension. This enables the assumption of a composite index in RDB. Since the performance highly depends on the selectivity of the first attribute, we set the store dimension as the first attribute. The second and third curves are Z-curves `(sdp)16(dp)16` and `(dp)16(sdp)16`. Since the bit length of the store dimension is shorter than those of the others, we prepared the two versions for evaluation. The last one is Curve `p8s6d12p10d20s10` designed with the proposed curve-design method. Since durations of day, week, and month have a fractional bit length, we rounded them up to apply the proposed method.

We also evaluated an R*-Tree implementation [2]. Note that we attempted to evaluate a Hilbert-curve-based index; however, it is too complex to implement a range-query processing algorithm for Hilbert curves. Furthermore, to the best of our knowledge, we could only find two available public implementations [21] and [3] of range query algorithms for Hilbert curves in dimensions larger than three. However, we could not apply them for our experiments because the former supports at most a 64-bit key length and the latter falls back to a full scan-based algorithm for larger query regions.

We implemented QUILTS using B-Tree to compare it with the R*-Tree implementation. We generated 100 M records based on the data model and set the page capacity to 1,000 to make each query access many more pages. The records were split into about 200,000 pages. We also generated 1,000 queries for each query pattern and evaluated the average number of page accesses per query.

The results are summarized in Figure 20. The vertical axis represents the average number of page accesses and is shown on a logarithmic scale. The horizontal axis represents query patterns.

The curve designed with the proposed method showed the best results for all query patterns. The curve reduced the number of accesses by at most one sixth compared to Z-curve 1, the second-best curve. Our curve was designed to fit as many query patterns as possible. On the other hand, the Z-curve was good at handling square-like query patterns but poor at handling elongated rectangle-like query patterns. Therefore, Z-curve 1 could not handle the query patterns in which the query-range-width of the product dimension was much wider than that of the time dimension. Z-curve 2 was much worse than Z-curve 1. The reason is that since the bit length of the store dimension is half those of the date and product dimensions, Z-curve 1 placed the bits of the store dimension at the least significant side and Z-curve 2 placed them at the most significant side. The results indicate that we should bit-interleave each dimension from the least significant side to handle different domain sizes.

The R*-Tree showed relatively good results for small query size. However, the R*-Tree accesses many pages for large query size. It splits pages in accordance with the data distribution. Since the data distribution of the product dimension is highly skewed, the R*-Tree tends to split data along the product dimension. This causes the R*-Tree to access many pages when the query-range width of the product dimension is wide. This indicates that to reduce the number of page accesses, we should consider the effect of query pattern more than that of data distribution. We can also see that the R*-Tree showed similar results to Z-curve 2. They show similar

Table 3: Aggregation Units and Query-Range Widths

Longitude & Latitude	Time
0.0001 (6.6 bits)	hour (11.8 bits)
0.001 (10.0 bits)	day (16.4 bits)
0.01 (13.3 bits)	week (19.2 bits)
0.1 (16.6 bits)	month (21.3 bits)

results on not only the average number of page accesses but also the number of page accesses per query.

6.3 GIS Application

We used trip record data of NYC yellow taxis from January to December 2015 [24]. The data set contains 150+ M records, but 2% of the data set is invalid due to the disorder of GPS devices. Each record contains pick-up or drop-off times and locations (latitude and longitude), number of passengers, fares, and so on. Figure 22 shows 1,000 sampled records. The data set is a typical skew-data distribution because there are one large cluster (Manhattan) and several small clusters (Broadway, JFK Airport, and LaGuardia Airport).

We assumed a spatio-temporal statistical analysis, such as the number of passengers or sales per region and/or duration, as a target workload. We considered the three dimensions of longitude, latitude, and time in the experiments. We assumed that the aggregation units for location are 0.0001, 0,001, 0.01, and 0.1 in decimal degree. These units are interpreted as individual street/land parcel ($7.871m^2$), neighborhood/street ($78.71m^2$), town/village (787.1 m^2), and large city/district ($7.871km^2$), respectively [1]. For the time dimension, we assumed four aggregation units: hour, day, week, and month. We considered 16 query patterns, by combining aggregation units in each dimension. Table 3 summarizes aggregation units in each dimension and query-range width in bit length. Note that (1) the location and time dimensions have different scales of query-range width in bit length and (2) aggregation units for the location dimension magnify in equal ratio but not for the time dimension.

We designed four curves `(txy)32`, `(xyt)32`, `(xy)5(txy)27t5`, and `(xy)5(xyt)10t2(xy)3t3(xy)4(xy)3t5(xyt)7t5`, where `t`, `x`, and `y` mean the time, longitude, and latitude dimensions, respectively. The first two curves are typical Z-curves. The first curve has time-precedence and the second one has location-precedence. Since the Z-curve is common for location data, these curves are baselines of performance comparison. The first curve designed with our curve-design method only took into account (1), and the second one took into account both (1) and (2). We also compared them with the R*-Tree. Each implementation holds about 200,000 pages. We generated 1,000 queries for each query pattern. The average number of page accesses are summarized in Figure 21.

Designed Curve 2 exhibited the best performance for ten query patterns due to the careful curve design. However, the curve performed worse, especially for query patterns whose width of the time dimension was short but those of the spatial dimensions were wide. As shown in Figure 17 in Section 5.3.2, our curve-design method could not support all query patterns. Since 4 query ranges per dimension were chosen in the experiments, there were 16 ($= 4 \times 4$) query patterns. However, the proposed method can fit at most 7 ($= 4 + 4 - 1$) query patterns because curves are defined by the order of bit-merging. Designed Curve 1 exhibited similar performance but slightly worse than Designed Curve 2. The two Z-curves performed well when the query patterns were square-like, 0.01 (13.3 bits) $\times$ hour (11.8 bits) and 0.1 (16.6 bits) $\times$ day (16.4 bits). The performance of the R*-Tree depended on the query range

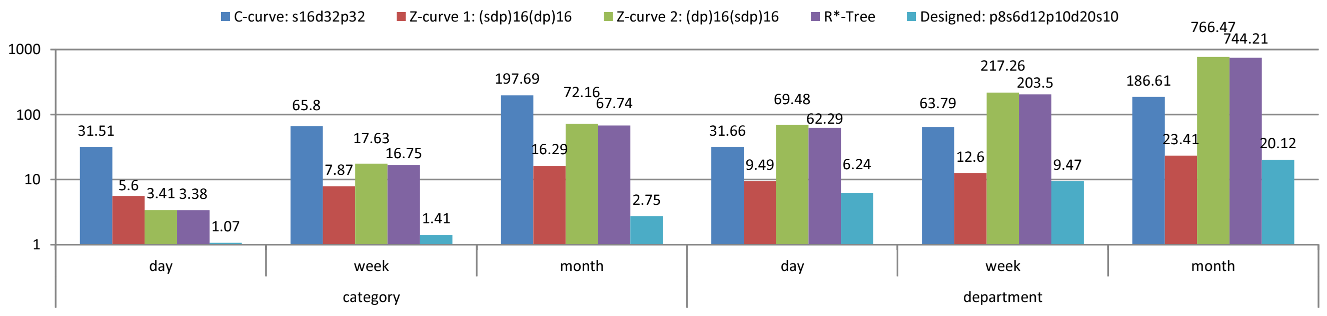


Figure 20: Average Number of Page Accesses by Retail Sales Analysis Query

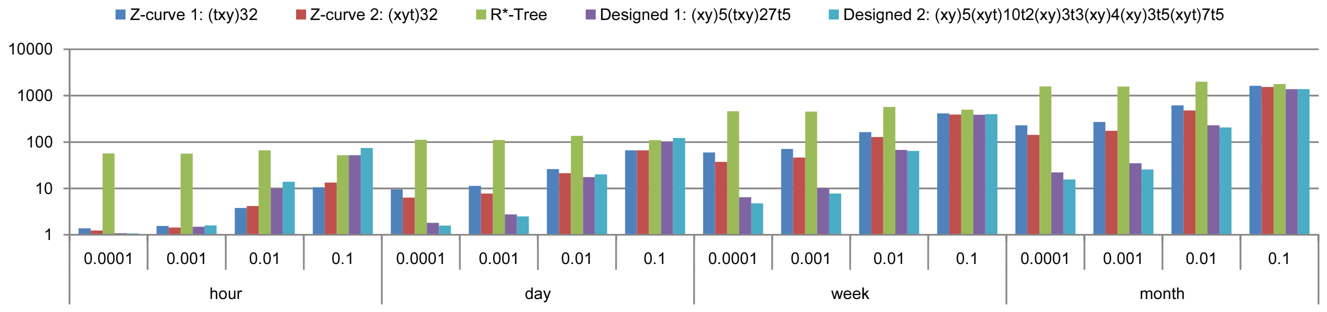


Figure 21: Average Numbers of Page Accesses through GIS Analysis Query

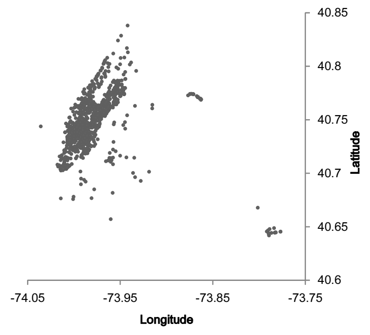


Figure 22: Spatial Data Distribution of NYC TAXI Data set

of the time dimension. All implementations showed similar results for the largest query pattern (0.1 for the spatial dimensions and month for the time dimension).

In summary, a well-designed curve can reduce the number of page accesses for targeting queries by an order of magnitude and treat the skewness of the data distribution. It is important to consider the order of bit-merging to fit target query patterns. However, there are limitations to fit any combination of query patterns by a single curve.

7. FUTURE WORK

7.1 Better Curve Exploration

The proposed curve-design method is a kind of heuristic. The designed curves performed well, but they do not limit the existence of better curves. However, since the bit-merging curve family contains a combinatorial explosive number of curves, an efficient candidate curve-exploration method is required. Currently, we do not have a closed form for cohesion cost calculation from a bit-merging order and a query pattern, the incurring cost to calculate it, especially for higher dimensional curves. One possible approach is to introduce the closed form for the cohesion cost calculated from a bit-merging order and target query pattern.

7.2 Curve Combination

There is a new opportunity to synthesize a hybrid curve other than the Z- and C-curves. One promising combination is the Hilbert curve and the U-curve [29, 14]. The Hilbert curve is expected to have lower local cohesion cost and higher global cohesion cost than the Z-curve. The U-curve is expected to have lower global cohesion cost and higher local cohesion cost than the C-curve. Therefore, the combination of the Hilbert curve and U-curve may cover the wider spectrum of the bit-merging curve family.

8. CONCLUSION

We proposed QUILTS, a framework of multidimensional data partitioning based on query-aware and skew-tolerant space-filling curves. QUILTS enabled us to improve query performance by using information about query patterns and query constraints. The technical contributions were: (1) a cohesion-based cost model that identifies the characteristics of a space-filling curve not only on a query pattern but also on data density, (2) a bit-merging curve family that enables us to synthesize a curve that inherits hybrid characteristics from the C- and Z-curves, and (3) a method for designing a query-aware and skew-tolerant curve to a given query pattern from the bit-merging curve. We prototyped the framework on top of HBase and the B-Tree, and conducted experiments. We confirmed that the framework reduces the amount of page accesses by an order of magnitude for DWH and GIS applications.

9. REFERENCES

[1] Decimal degree. https://en.wikipedia.org/wiki/Decimal_degrees.

[2] libspatialindex. https://github.com/libspatialindex/libspatialindex.

[3] uzaygezen. https://github.com/aioaneid/uzaygezen.

[4] M. Bader. *Space-Filling Curves: An Introduction with Applications in Scientific Computing*, volume 9 of *Texts in Computational Science and Engineering*. Springer Berlin Heidelberg, 2013.

[5] C. Faloutsos. Gray codes for partial match and range queries. *IEEE Transactions on Software Engineering*, 14(10):1381–1393, Oct. 1988.

[6] C. Faloutsos. Multiattribute hashing using gray codes. In *the ACM SIGMOD Conference*, pages 227–238, May 1986.

[7] C. H. Hamilton and A. Rau-Chaplin. Compact hilbert indices: Space-filling curves for domains with unequal side lengths. *Information Processing Letters*, 105:155–163, 2008.

[8] M. Hazewinkel, editor. *Encyclopedia of Mathematics*, chapter Multinomial coefficient. Springer, 2001. http://www.encyclopediaofmath.org/index.php/Multinomial_coefficient.

[9] HBase: Bigtable-like structured storage for Hadoop HDFS, 2010. http://hadoop.apache.org/hbase/.

[10] D. Hilbert. Ueber stetige abbildung einer linie auf flächenstück. *Mathematische Annalen*, 38:459–460, 1891.

[11] S. Huang, B. Wang, J. Zhu, G. Wang, and G. Yu. R-hbase: A multi-dimensional indexing framework for cloud computing environment. In *Data Mining Workshop (ICDMW), 2014 IEEE International Conference on*, pages 569–574, Dec 2014.

[12] R. Kimball and M. Ross. *The Data Warehouse Toolkit: the complete guide to dimensional modeling*. Wiley Computer Publishing, second edition, 2002.

[13] J. K. Lawder. Querying multi-dimensional data indexed using the hilbert space-filling curve. *SIGMOD Record*, 30:2001, 2001.

[14] X. Liu and G. F. Schrack. A new ordering strategy applied to spatial data processing. *International Journal Geographical Information Science*, 12(1):3–22, Jan. 1998.

[15] V. Markl. *MISTRAL: Processing Relational Queries using a Multidimensional Access Technique*. PhD thesis, TU München, 1999.

[16] V. Markl and R. Bayer. Processing Relational OLAP Queries with UB-Trees and Multidimensional Hierarchical Clustering. *Proceedings of the International Workshop on Design and Management of Data Warehouses*, 2000:1–10, 2000.

[17] M. F. Mokbel and W. G. Aref. Irregularity in multidimensional space-filling curves with applications in multimedia databases. In *In Proceedings of the International Conference on Information and Knowledge Managemen, CIKM*, 2001.

[18] M. F. Mokbel and W. G. Aref. On query processing and optimality using spectral locality-preserving mappings. *Advances in Spatial and Temporal Databases Lecture Notes in Computer Science*, 2750:102–121, 2003.

[19] M. F. Mokbel, W. G. Aref, and I. Kamel. Analysis of multi-dimensional space-filling curves. *Geoinformatica*, 7(3):179–209, Sept. 2003.

[20] B. Moon, H. V. Jagadish, C. Faloutsos, and J. Salz. Analysis of the clustering properties of hilbert space-filling curve. *IEEE Trans. Knowl. Data Eng., TKDE*, 13(1):124–141, 2001.

[21] D. Moore. Fast hilbert curve generation, sorting, and range queries. http://www.tiac.net/~sw/2008/10/Hilbert/moore/index.html.

[22] G. M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, IBM Ltd., 1966.

[23] S. Nishimura, S. Das, D. Agrawal, and A. El Abbadi. MD-HBase: A Scalable Multi-dimensional Data Infrastructure for Location Aware Services. In *Proceedings of 12th IEEE International Conference on Mobile Data Management, MDM*, pages 7–16, 2011.

[24] NYC Taxi & Limousine Commission. TLC Trip Record Data. http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.

[25] V. Raman, G. Attaluri, R. Barber, N. Chainani, D. Kalmuk, V. KulandaiSamy, J. Leenstra, S. Lightstone, S. Liu, G. M. Lohman, T. Malkemus, R. Mueller, I. Pandis, B. Schiefer, D. Sharpe, R. Sidle, A. Storm, and L. Zhang. Db2 with blu acceleration: So much more than just a column store. *Proc. VLDB Endow.*, 6(11):1080–1091, Aug. 2013.

[26] F. Ramsak, V. Markl, R. Fenk, M. Zirkel, K. Elhardt, and R. Bayer. Integrating the ub-tree into a database system kernel. In *26th International Conference on Very Large Data Bases*, pages 263–272, Sep. 2000.

[27] H. Sagan. *Space-Filling Curves*. Springer-Verlag, 1994.

[28] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[29] G. Schrack and X. Liu. The spatial u-order and some of its mathematical characteristics. In *the Pacific Rim Conference on Cummunications, Computers, and Signal Processing*, pages 416–419, May 1995.

[30] T. Skopal, M. Krátký, J. s. Pokorný, and V. Snášel. A new range query algorithm for Universal B-trees. *Information Systems*, 31(6):489–511, Sept. 2006.

[31] L. Sun, M. J. Franklin, S. Krishnan, and R. S. Xin. Fine-grained partitioning for aggressive data skipping. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 1115–1126, New York, NY, USA, 2014. ACM.

[32] Sybase, Inc. *Performance and Tuning: Basics*, Aug. 2003. Chapter 13: Indexing for Performance.

[33] H. Tropf and H. Herzong. Multidimensional range search in dynamically balanced trees. *Angewandte Informatik*, 23(2):71–77, Feb. 1981.

[34] M. White. N-trees: large ordered indexes for multi-dimensional space. Technical report, Statistical Research Division, US Bureau of the Census, 1982.

[35] P. Xu and S. Tirthapura. On the optimality of clustering properties of space filling curves. In *Proceedings of the 31st Symposium on Principles of Database Systems*, PODS '12, pages 215–224, New York, NY, USA, 2012. ACM.

[36] Y. Zou, J. Liu, S. Wang, L. Zha, and Z. Xu. Ccindex: A complemental clustering index on distributed ordered tables for multi-dimensional range queries. *Network and Parallel Computing*, 6289:247–261, 2010.