Fractal Dimension and Space Filling Curve

Ana Karina Gomes
anakgomes@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

May 2015

Abstract

Space-filling curves are computer generated fractals that can be used to index low-dimensional spaces. There are previous studies using them as an access method in these scenarios, although they are very conservative only applying them up to four-dimensional spaces. Additionally, the alternative access methods tend to present worse performance than a linear search when the spaces surpass the ten dimensions. Therefore, in the context of my thesis, I study the space-filling curves and their properties as well as challenges presented by multidimensional data. I propose their use, specifically the Hilbert curve, as an access method for indexing multidimensional points up to thirty dimensions. I start by mapping the points to the Hilbert curve generating their h-values. Then, I develop three heuristics to search for approximate nearest neighbors of a given query point with the aim of testing the performance of the curve. Two of the heuristics use the curve as a direct access method and the other uses the curve as a secondary key retrieval combined with a B-tree variant. These result from an iterative process that basically consists of planning, conceiving and testing the heuristic. According to the test results, the heuristic is adjusted or a new one is created. Experimental results with the three heuristics prove that the Hilbert curve can be used as an access method, and that it can operate at least in spaces up to thirty-six dimensions.

Keywords: Fractals, Hilbert Space-Filling Curve, Low-Dimensional Indexing, Approximate Nearest Neighbor

1. Introduction

Data is becoming increasingly complex. It contains an extensive set of attributes and sometimes requires to be seen as part of a multidimensional space. Database Management Systems (DBMSs) have undergone several changes in its access methods in order to support this type of data. Throughout the years, several indexes have been proposed to handle multidimensional (or spatial) data, but the most popular is the R-tree. Despite all the effort dedicated to improve it, R-tree performance does not appear to keep up the increasing of the dimensions. It is common to multidimensional indexes to suffer from the *curse of dimensionality* which means they present a similarity search performance worse than a linear search in spaces that exceed ten dimensions [18, 24, 28]. Even with other indexes with better performance, Mamoulis considers that similarity search in these spaces is a difficult and unsolved problem in its generic form [18].

The approximate nearest neighbor is a similarity search technique that neglect the accuracy of query results for reduced response time and memory usage. It is an invaluable search technique in high dimensional spaces. Commonly, we use feature vectors to simplify and represent data in order to lower the complexity. The features are represented by points in the high dimensional space and the closer the points in space, the more similar the objects are. In the case of images, these are usually represented by feature vectors, which are then used to find approximately similar objects. In this case, an approximate result is an acceptable solution. Multimedia data is only an example among several other applications. The approximate nearest neighbor, as its name suggests, does not guarantee the return of the exact neighbor, and its performance is highly related to the data distribution [27]

Ideally, in order to reduce the search time, all the multidimensional data that are related should be linearly stored on disk or memory. However, this cannot be done because it is impossible to guarantee that all the pair of objects close in space will be all close in the linear order. It is impossible to all but not for some, and the space-filling curves provide a total order of the points in space with superior clustering property. Another problem relating the multidimensional data refers to its indexing structures. Although, there are several low-dimensional indexing structures, they present a problem that concerns

with the number of dimensions increase. When this happens, the volume of the space increases so fast that the data in the space become sparse. The curse of dimensionality appears when the spaces usually surpass the ten dimensions, comparing their performance to a linear search or worse [8]. Berchtold et al. refer that they also suffer from the well-known drawbacks of multidimensional index structures such as high costs for insert and delete operations and a poor support of concurrency control and recovery. A possible solution to this problem is to use dimensionality reduction techniques like the space-filling curves [3, 8]. They allow to map a d-dimensional space to a one-dimensional space in order to reduce the complexity. After this, it is possible to use a B-tree variant to store the data and take advantage of all the properties of these structures such as fast insert, update and delete operations, good concurrency control and recovery, easy implementation and re-usage of the B-tree implementation. Additionally, the space-filling curve can be easily implemented on top of an existing DBMS [3].

For all the reasons mentioned above, the purpose of this paper is to study the space-filling curves and their properties as well as challenges presented by multidimensional data. I explore the use of the space-filling curves, specifically the Hilbert curve, as an access method for indexing multidimensional points. I also test three heuristics to search for an approximate nearest neighbor on the Hilbert curve. The main goal is to explore the potential of the Hilbert curve in low dimensional spaces (up to thirty dimensions). These spaces may result from the application of indexing structures oriented to high-dimensional space or result from the application of dimensionality reduction techniques for subsequent indexing. In either situation, the resulting space, despite being of lower dimensions, still requires a large computational effort to perform operations on data.

2. Multidimensional Indexing

There are several structures oriented for indexing multidimensional data. These vary according to their structure, if they have taken into account or not the data distribution, if they are targeted for points or regions, if they are more suited to high or low dimensional spaces, etc. For example, structures such as Kd-trees [1, 2], hB-trees [17], Grid files [20], Point Quad trees among others, are oriented to indexing points. Structures such as the Region Quad trees [12], SKD [21] or R trees can handle both points or regions [22]. Subspace-tree [25] and LSD-tree are more suited to high dimensions and R-tree, Kd-tree and Space-Filling Curves are more suited do low dimensional spaces. Theses lists are quite incomplete due to the wide variety of existing proposals. Although there is no consensus on the best structure for multidimensional indexing, the R tree is the most commonly used for benchmark the performance of new proposals. Furthermore, in commercial DBMSs, the R-trees are preferred due to their suitability for points and regions and orientation to low-dimensional space [28, 22]. The list of alternative access methods to compete with fractals in low-dimensional space (two to thirty dimensions) are quite extensive. Therefore, I choose the R-tree and the Kd-tree as two possible alternatives to take a closer look.

2.1. R-tree

Since the multidimensional indexes are primarily conceived for secondary storage, the notion of data page is a concern. These access methods perform a hierarchical clustering of the data space, and the clusters are usually covered by page regions. To each data page is assigned a page region which is a subset of the data space. The page regions vary according to the index, and in the R-tree case has the name of minimum bounding rectangle (MBR) [5]. At a higher-resolution level, each MBR represents a single object and is stored on the lower leaf nodes. These leafs contain also an identifier of a multidimensional point or region that represents some data. The R-tree descends from the B+-tree and therefore, preserves the height-balanced property. Thus, at the higher levels will have cluster of objects, and since they are represented by rectangles, we will also have clusters of rectangles [28, 22].

The R-tree search performance relates to the size of the MBRs. Note that if there are too many overlapping regions, when performing a search it will result in several intersections with several subtrees of a node. This will require to traverse them all, and we must be considering that the search occurs in a multidimensional space. Thus, the minimization of both coverage and overlap, has a large impact on the performance [28, 22]. Other proposals based on R tree have been presented over the years. The R*-tree tries to improve the R-tree by choosing coverage of a square shape rather than the traditional rectangle. This shape allows a reduction in coverage area by reducing the overlap and hence improving performance of the tree. Like the R*-tree, many other structures have been proposed based on the R-tree as the R+-tree, the X-tree, the Hilbert R-tree among others. However, there is also another problem that haunts these hierarchical indexing structures known as the *curse of dimensionality*. The main underlying cause relates to the volume of the chosen form that covers the points. This has a constant radius or edge size which increases exponentially with the increasing of dimensions [4, 26]. Nevertheless, the R-tree is the most popular multidimensional access method and has been used as a

benchmark to evaluate the new structures' proposals [28, 22].

2.2. Kd-tree

Throughout the years, several hierarchal indexes have been proposed to handle multidimensional data space. These structures evolve from the basic B+ tree index and can be grouped in two classes: indexes based on the R-tree and indexes based on the K-dimensional tree (Kd-trees). Unlike the R-tree, the Kd-tree is unbalance, and point and memory-oriented . It divides the data space through hyper-rectangles resulting in mutually disjoint subsets. Like the R-tree, the hyper-rectangles correspond to the page regions but in this case, most of these do not represent any object [5, 26]. If a point is on the left of the hyper-rectangle it will be represented on the left side of the tree, otherwise it will be represented on the right side. The advantage is that the choice of which subtree to search is always unambiguous [1, 28].

There are some disadvantages of having complete disjoint partitions of the data space. Bohm et al. refer that the page regions are normally larger than necessary, which can lead to an unbalanced use of data space resulting in high accesses than with the MBRs [5]. The R-tree only covers the space with data unlike the Kd-tree. The latter is unbalanced by definition, in a way that there are no direct connections between the subtrees structure [28]. This leads to the impossibility of pack contiguous subtrees into directory pages. The Kd-B-tree [23] results from a combination of the Kd-tree with the B-tree [9] and it solves this problem by forcing the splits [5, 18]. This culminates in a balancer resulting tree, such as the B-tree, overcoming the problem above. Creating completely partitions has its drawbacks. Namely, the rectangles can be too big containing only a cluster of points. Or the reverse, a rectangle may have too many points becoming overloaded. Thus, more adjustable rectangles may bring better performance. The hB-tree tries to overcome this problem decreasing the partition area using holey bricks. However, the hB-tree is not ideal for disk-based operation [28]. Berchtold et al. refer that all these index structures are restricted with respect to the data space partitioning. They also suffer from the well-known drawbacks of multidimensional index structures such as high costs for insert and delete operations, and a poor support of concurrency control and recovery [3].

3. Fractals

A fractal can be found in Nature or can be computer generated thanks to its self-similarity property. A space-filling curve is an example of computer generated fractal. Jordan, in 1887, defines formally a curve as a continuous function with endpoints whose domain is the unit interval [0,1] [19, 13] but the space-filling curve is more than a simple curve, it has special properties. It consists on a path in multidimensional space that visits each point exactly once without crossing itself, hence introducing a total ordering on the points. However, there is no total order that fully preserves the spatial proximity between the multidimensional points in order to provide a theoretically optimal performance for spatial queries. According to Moon et al., once the problem of the total order is overcome, it is possible to use the traditional one-dimensional indexing structures for multidimensional data with good performance on similarity queries [19].

3.1. Hilbert Curve

The Hilbert Curve is a space-filling curve based on Peano Curve and comes as an improvement [10]. The Peano basic curve has the shape of an upside down "U". The curves of higher order are obtained by replacing the lower vertices for the previous order curve. The upper vertices are also replaced by the previous order curve but suffering a rotation of 180 degrees. The resulting curves can be seen in [10]. The Hilbert starts with the same Peano's basic curve but evolves differently. On the following orders, the top vertices are replaced by the previous order, and the bottom vertices suffer a rotation (see Figure 1). The bottom left vertex is rotated 90 degrees clockwise, and the bottom right rotates 90 degrees counterclockwise. The Figure 1 shows the Hilbert Curve orders one, two and three. In this, the curve starts on the lower left corner and ends on the lower-right corner, but this can be changed as long as the curve keeps the "U" shape.

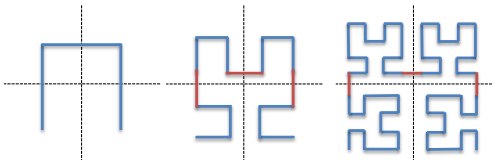


Figure 1: Hilbert Curve orders 1, 2, and 3 respectively.

3.2. Higher Dimensions

We have seen how the Hilbert space-filling curve works in 2 dimensions. Now, we will see the behavior of the Hilbert curve in when the dimensions are bigger than 2. The concepts introduced here are based on [14]. The author does a geometric approximation to the curve in order to extend the concept to higher dimensions. We are focusing on the Hilbert curve since it generally presents the best cluster property, which translates to a reduction in disk (or memory) accesses required to perform a

range query [19]. To simplify the explanation, the author chose to use the Hilbert first-order curve rotated 90 degrees clockwise and then 180 degrees along the horizontal axis (see Figure 2).

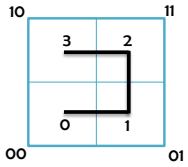


Figure 2: Hilbert curve generic analysis

In order to increase the curve dimensions, we need to generalize its construction and its representation. If we look closely to the Hilbert first order curve in two dimensions (see Figure 1), we can observe an unfinished square with 2×2 vertices. Each one of the them represents a single point. On the second-order curve, we have $2^2 \times 2^2$ vertices due to the recursive construction of the curve. As the order curve increases, we will have $2 \times \cdots \times 2$ points in n dimensions corresponding to the hypercube vertices. Each of the 2^n vertices are represented by a n-bit string such as

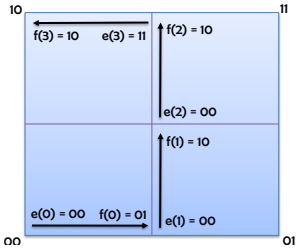
$$b = [\beta_{n-1} \cdots \beta_0] \tag{1}$$

where $\beta_i \in \mathbb{B}$ takes 0 (low) or 1 (high) according to its position on the grid. Looking at the Figure 2, we can conclude that $b = [00]$ corresponds to the lower left vertex of the square and $b = [11]$ corresponds to the top right vertex of the square. Along the curve, all the vertices are immediate neighbors. This means that their binary representation only changes in one bit. The directions of the curve inside the grid are therefore represented using the Gray code. The function $\text{gc}(i)$ returns the Gray code value for the ith vertex. The symbols $\oplus$ and $>>$ represent the exclusive-or and the logical shift right respectively.

$$\text{gc}(i) = i \oplus (i >> 1) \tag{2}$$

The curve is constructed by replacing each vertex for the previous order curve. Each replacement is a transformation/rotation that generates a new sub-hypercube. These operations must make sense along the 2^n vertices. With sense, I mean that "every exit point of the curve through one sub-hypercube must be immediately adjacent to the entry point of the next sub-hypercube" [14]. The author defines an entry point $e(i)$ where i refer to the ith sub-hypercube.

$$e(i) = \begin{cases} 0, & i = 0 \\ \text{gc}(2\left\lfloor \frac{i-1}{2} \right\rfloor), & 0 < i \leq 2^n - 1 \end{cases} \tag{3}$$



i	$e(i)$	$f(i)$	$d(i)$	$g(i)$
0	$[00]_2$	$[01]_2$	0	0
1	$[00]_2$	$[10]_2$	1	1
2	$[00]_2$	$[10]_2$	1	0
3	$[11]_2$	$[10]_2$	0	

Figure 3: Analysis of the entry and exit points of Hilbert first-order curve in two dimensions. The author refer that the x corresponds to the least significant bit and the y to the most significant. Adapted from [14]

The entry point $e(i)$ and the exit point $f(i)$ are symmetric and the latter is defined as

$$f(i) = e(2^n - 1 - i) \oplus 2^{n-1} \tag{4}$$

Even at the first-order curve, we can look at each vertex as sub-hypercube. Therefore, each one as an entry and exit point. We can look to the Figure 3 and observe that to generate the first vertex/sub-hypercube the curve entry on the left lower corner (00) and exit at the right lower corner (01). The next vertex is generated by entering on the left lower corner (00) and exiting on the right upper corner (10). And so on. Additionally, we need to know along which coordinate is our next neighbor in the curve. The directions between the sub-hypercubes are given by the function $g(i)$ where i refer to the ith sub-hypercube.

$$g(i) = k \text{ such that } \text{gc}(i) \oplus gc(i+1) = 2^k \tag{5}$$

The author refers that $g(i)$ can also be calculated as $g(i) = tsb(i)$ where tsb is the number of trailing set bits in the bynary representation of i. The $g(i)$ indicate the *inter* sub-hypercube direction along the curve. We can also calculate *intra* direction $d(i)$ on the sub-hypercubes.

$$d(i) = \begin{cases} 0, & i = 0; \\ g(i-1) \mod n, & i = 0 \mod 2; \\ g(i) \mod n, & i = 1 \mod 2, \end{cases} \tag{6}$$

$for\ 0 \leq i \leq 2^n - 1.$

All these functions can be calculated for any dimension, allowing to construct a Hilbert curve of higher dimensions. The author main idea is to define a geometric transformation T such that the ordering of the sub-hypercubes in the Hilbert curve defined by e and d will map to the binary reflected Gray code" [14]. The operator $x \circlearrowright i$ is defined as the right bit rotation and rotate n bits of x to the right i places. A more formal definition is given in [14].

$$T_{e,d}(b) = b \oplus e \circlearrowright (d+1) \tag{7}$$

3.3. Fractals: An Access Method

The use of space-filling curves for indexing multidimensional spaces is not something new. Basically, there are two ways to use them and perform queries on space-filling curves. One hypothesis is to use a variant of B-tree combined with the curve as a secondary key retrieval. The other alternative is to use the curve directly through the h-values. In this section, we will see how these indexes are used in the related work. Since the scope of this work is also focused on similarity search, I will only present works underlying this matter or showing different ways to use fractals as an access method.

3.3.1 Secondary Key Retrieval

Faloutsos et al. in [11] propose the use of fractals to achieve good clustering results for secondary key retrieval. The performance of an access method for secondary key depends on how good is the map to preserve locality. The authors proceed with the study on the basis that the Hilbert curve shows a better locality preservation map by avoiding long jumps between points. To a more interesting study, it is added two other curves to compare with the Hilbert curve, the Z-curve and the Gray-code curve. The analysis focus on range and nearest neighbor queries to test the hypothesis. The main purpose of their study is to find the best locality preserving map. The best is defined based on two measures. The first focus on the performance of the locality preserving map when using range queries. Simplifying, the average number of disk accesses required for a range query that is exclusively dependent on the map, revealing how good the clustering property is. The second measure is related with the nearest neighbor query, and its called the maximum neighbor distance. Faloutsos et al. refer that for a given locality preserving mapping and a given radius R, the maximum neighbor distance of a point X is the largest Manhattan - L_1 metric - distance (in the k-dimensional space) of the points within distance R on the linear curve. The shorter the distance between the points the better the locality preserving map. In order to make the experiment feasible, the maps are tested for dimensions two, three and four to fourth order. Generally, the Hilbert curve presents better results than the other two curves for the two measures applied. This study thus contributes to encourage the use of fractals as secondary keys, especially the Hilbert curve. However, despite the positive contribution this study makes, it would be interesting to know the performance of fractal beyond the four dimension.

Lawder et al. introduce a more developed work based on this proposal. They present a multidimensional index based on the Hilbert curve. The

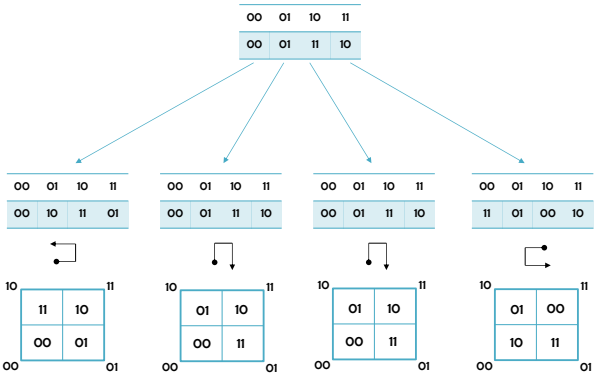


Figure 4: Indexing structure based on the Hilbert curve - Example of a indexing a second order curve. Adapted from [16].

points are partitioned into sections according to the curve, and each section corresponds to a page storage. The h-value of the first point in a section is used as corresponding page, given to the pages a coherent ordering. The page-keys are then indexed using a B-tree variant. The height of the tree reflects the order of the curve indexed, and the root corresponds to the first-order curve. Each the node makes the correspondence between the h-value and the position of the point on the grid (see Figure 4). For the second-order curve level, the first ordered pairs are the parent of the lower nodes, and so on. In the Figure 4, I present an example adapted from [16] that represent an index of a Hilbert second order curve. The nodes have a top line and a bottom line. The top one corresponds to the ith vertex of the curve. And the bottom one corresponds to its position on the grid, like we explained in Section 3.2. Reading the bottom line of the node from left to right, we describe the curve's path on the grid. So, the first (left) node, in the second level of the hierarchy, the starts on the left lower corner (00), to the right lower corner (01), to the righ upper corner (11) and finally to the left upper corner (10). The same are valid for the rest of the nodes. Since the tree can easily become too big, the authors use a state diagram, suggest by [11], to represent the four possible nodes and express the tree in a compact mode. However, this only works up to 10 dimensions. Above that the "memory requirements become prohibitive"[16]. Some modifications are made but the method only works up to 16 dimensions. The implementation of the tree is compared with the R tree, revealing that indexing based on the Hilbert curve saved 75% of time to populate the data store and reduced by 90% the time to perform range queries. The authors focused on points as datum-points (records) and a further investiga-

tion considering points as spatial data would also be interesting. To more information on Lawder et al. indexing structure, please see [16].

3.3.2 Direct Access Index

Another interesting indexing structure proposal based on the Hilbert curve is from Hamilton et al. in [15]. They present an algorithm based on [6] and perform a geometrical approximation of the curve already presented in Section 3.2. The authors developed a compact hilbert index that enables a grid to have different sizes along the dimensions. The index simply converts a certain point p to a compact h-value. The authors extract redundant bits that are not used to calculated the index. This allows a reduction in terms of space and time usage. The index is tested and compared with a regular Hilbert index and Butz's algorithm. The authors conclude that although the compact Hilbert index is more "computationally expensive to derive", it saves significantly more space and reduces the sorting time. The compact Hilbert index is tested up to 120 dimensions to map the indexes [15].

Chen et al. also choose to use the Hilbert curve using the h-values for a direct access to the points. They perform a set of operations similar to what we present in Section 3.2 in order to access a certain point. Unlike Hamilton et al., they also developed a query technique based on the map. There are two basic steps to find the nearest neighbor to a query point. First, locate the query point on the curve, then locate the neighbor. To locate a certain point on the grid, they start by defining the direction sequence of the curve (DSQ) based on the cardinal points. The direction is fixed starting from left lower vertex (SW) and finishing at the right lower vertex (SE). The DSQ is represented as (SW,NW,NE,SE). Each of the four cardinal points is then substitute for the h-value at that position. The combination of the h-values with the cardinal points gives us the direction of the curve in each sub-hypercube. The h-value of the query point is converted to a number of base four $[d_1 \cdots d_m]_4$ and each digit indicates the curve's direction along the zoom in on the curve. Or in other words, the number of digits m refers to the order of the curve. Additionally, as we know, the curve suffers some transformations/rotations along the sub-hypercubes. These transformations are here defined and related with the derived quaternary number composed along zoom in:

- **C13** - If the number ends with zero, then switch all ones for three and vice-versa;
- **C02** - If the number ends with three, then switch all zeros for two and vice-versa;
- **C11/22** - If it ends with one or two, remains the same.

The authors tested their algorithm with six distinct spatial datasets and compared with a previous version that computes the exact location through the binary bits of the coordinates. The last version showed better results despite only being tested in two dimensions [7].

4. Approximate Space-Filling Curve

In this paper, I proposed to study the problem of index low-dimensional data up to thirty dimensions, using the Hilbert space-filling curve as a dimensionality reduction technique that is known to have superior clustering properties. To address the problem, I create a framework named Approximate Space-Filling Curve that structured all the steps of the proposal.

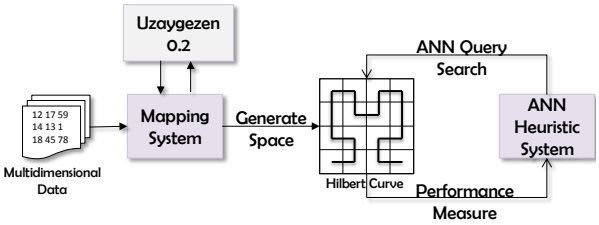


Figure 5: Approximate Space-filling Curve Framework

4.1. Proposal

This framework has two main phases: the Mapping System and the Approximate Nearest Neighbor Heuristic System (ANN Heuristic System). The first system converts each one of the multidimensional point in the dataset to the Hilbert curve, generating its h-value. Each point is mapped with the auxiliary of the Uzaygezen 0.2 library. It is based on the theory of the Compact Hilbert Indexes [14] already presented in the Section 3.2 and it follows this basic algorithm [15]:

1. Locate the cell that contains our point of interest by determining whether it lies in the upper or lower half-plane regarding each dimension. Assuming we are working with a m-order curve where m also refers to the number of bits necessary to represent a coordinate,

$$l_{m-1} = [\text{bit}(p_{n-1}, m-1) \ldots \text{bit}(p_0, m-1)] \quad (8)$$

2. We zoom in the cell containg the point and we rotate and reflect the space such that the Gray code ordering corresponds to the binary reflected Gray code. The goal is to transform the curve into the canonical orientation.

$$\bar{l}_{m-1} = T_{e,d}(l_{m-1}) \quad (9)$$

3. The orientation of the curve is given only by an entry e and the exit f through the grid. Once we know the orientation at the current resolution, we can determine in which order the cells are visited. So, h gives the index of a cell at the current resolution.

$$h = [w_{m-1} \cdots w_0] \tag{10}$$

Where w is the index of the associated cell.

4. We continue until sufficient precision has been attained.

On the second system, I develop a heuristic, test on the curve and re-adapt it based on its performance. As a result of this process, three heuristics were created: the Hypercube Zoom Out (HZO), the Enzo Full Space (FS) and finally the Enzo Reduced Space (RS).

The HZO searches for the approximate neighbor starting at the highest resolution hypercube excluding the query point itself. It starts at the entry point of the hypercube and ends the first iteration at the end point of the same hypercube. If it does not find any neighbor, it performs a zoom out to the hypercube of the next resolution. Always starting at an entry point and ending on an exit point. Once it finds the first neighbor, the heuristic ends. The Enzo FS searches for the approximate neighbor by launching two search branches from the query point to lower resolution entry an exit point. If both branches find neighbors, they compare them and the closest neighbor to the query point is the approximate nearest neighbor.

Unlike the previous two which use the Hilbert curve as a direct access, the Enzo RS uses the Hilbert curve as a secondary key retrieval combined with a B-tree variant. Instead of generating neighbors and verifying if they exist in the data, it uses the data ordered according to the curve. Using the B-tree variant and the h-value of the query point, it is determined the query point immediate neighbors on the curve. Only later, when writing the report and after tested the heuristics, I could notice that the Faloutsos et al. had already suggested this heuristic in [11]. Their algorithm was therefore presented in Section 3.3 as related work. Nevertheless, they only tested the heuristic till 4 dimensions.

4.2. Experiments

The code was developed in JAVA and tested in a computer with the following properties:

- Windows 8 Professional 64 Bits
- Processor Pentium(R) Dual-Core CPU T4300 @ 2.10GHz
- 4 GB RAM

The dataset is a result from applying a Subspace tree [26] to high-dimensional data. A few experiments were performed with the resulting prototype in six spaces with $2, 4, 5, 10, 30$ and 36 dimensions respectively.

The datasets are organized in files per dimension. The dimensions tested are 2, 4, 5, 10, 30 and 36. The spaces with dimensions 2, 4 and 36 have 10.000 points and the remain have 100.000. The files are analyzed and duplicated data are removed. After this cleaning process, the files have the number of points presented in Table 1, in the column "Data Volume". It is possible to know already the order of the Hilbert curve through the number of bits to represent the higher number present in a file. The order of the curve is therefore, in this same table, in the column labeled as "Resolution".

Spaces	Original Volume	Data Volume	Resolution
S2	10 000	264	6
S4	10 000	9 229	7
S5	100 000	98 917	16
S10	100 000	98 917	16
S30	100 000	98 917	16
S36	9 877	9 831	8

Table 1: Datasets description

Before running the heuristics experiments, I defined the set of query points for which I intend to find their neighbors. I defined as well the distance values of their nearest neighbors with a linear search. The query group vary from de point $P0$ to $P9$ and are the first 10 points extracted from each file after the cleaning process. The spaces of dimension 2 and 4 are densely populated since all the neighbors are immediate cell neighbors, and the mean distance is 1. The spaces regarding dimensions 5, 10 and 30 are quite sparse since the distance mean are $332,3$, $629,2$ and $943,4$ respectively. Finally, for the space 36 we can say that it is in between the first two and the rest of the spaces regarding the density of the data. It presents a mean distance of $34,7$. In terms of analysis, I grouped the spaces in two sets like {2, 4, 36} and {5, 10, 30} because they are more similar between them as shown above.

The heuristics were tested considering the runtime performance and the relative error of the approximate nearest neighbor. Once tested, both HZO and Enzo FS did not present any results in a time considered acceptable (less than a day) in spaces with more than 4 dimensions. They revealed to produce huge amounts of data since they generate the h-value of the potential neighbor and verify if it exists on the dataset. In the worst-case scenario, the space 30 at the order curve 16, they can generate the number of cells existent in

the maximum order curve, which is an amazing $3,121749 \times 10^{144}$ cells. The heuristics required high computational resources, namely space, and due to the space constraint, those results could not be computed.

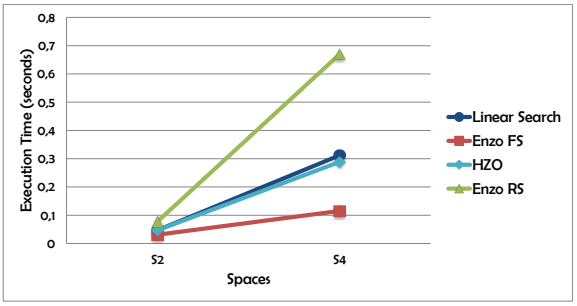


Figure 6: Runtime comparison between the heuristics.

Nevertheless, in the spaces 2 and 4, regarding the runtime performance, the HZO revealed to be a slightly better than a linear search and the Enzo FS slightly better than HZO (Figure 6). Although, we must take into account that the results are approximate. In terms of the overall approximate error distance, 85% of the both heuristics results are the nearest neighbor to a given query point. Globally, the average relative error is $0,35$ cells away from the nearest neighbor for the HZO and $0,25$ for the Enzo FS (Figure 7).

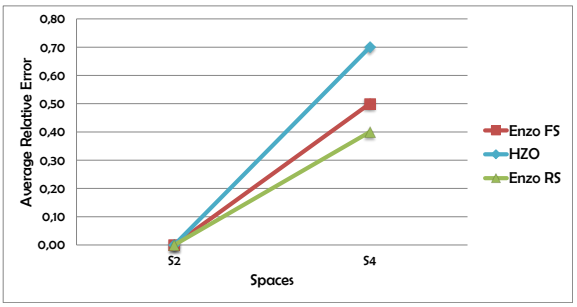


Figure 7: Average relative error comparison between the heuristics.

The Enzo RS, unlike the previous two, presented results for all the dimensions tested (2, 4, 5, 10, 30 and 36). On what concerns the runtime performance, it revealed to be worse than the linear search in spaces till 4 dimensions. On what concerns the error of the approximate neighbors distance in these spaces, the Enzo RS presented an average relative error of 0.2 (Figure 7). Therefore, this heuristic presented the lowest average approximate error but the higher runtime performance in these spaces. Beyond these spaces, the Enzo RS worked in less time than the linear search in all dimensions tested (Figure 8). The spaces in the Figure 8 are sorted in ascending order of resolution, the amount of data to compute, dimensions and average distance neighbor. It is possible to observe that the Enzo RS, is faster than the linear search from the space 36 although it starts to stand out from the space 10. The lower the density of the area, the greater the distance between the Enzo RS curve and the linear search curve. Never forgetting that we are talking about approximate neighbors, and the linear search always provides us the nearest.

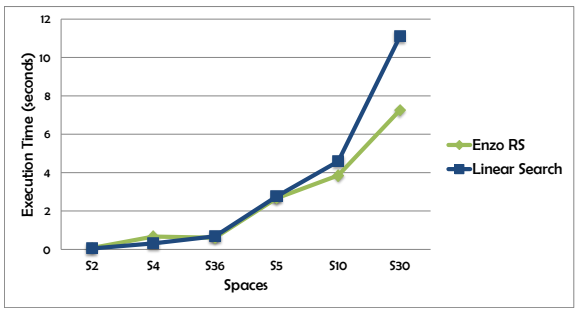


Figure 8: Enzo RS runtime performance along the spaces.

The average relative error of the Enzo RS seems to grow with the increase of the dimensions (Figure 9). Globally, on all the query points tested, the Enzo RS presented an average relative error of $0,61$ cells away from the nearest neighbor (Figure 9). Despite the relative error is higher than the previous heuristics, the reader should not forget that this heuristic was tested in more forty queries than the previous. So we cannot compare the overall relative error.

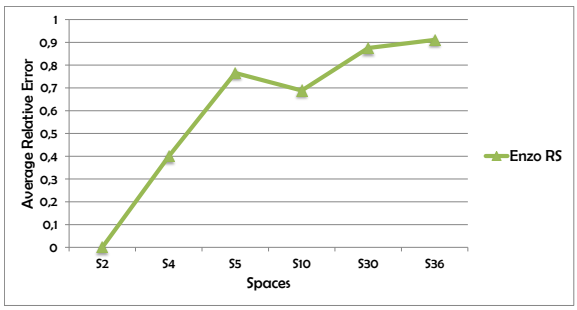


Figure 9: Enzo RS average relative error along the spaces.

Overall, in the spaces 2 and 4, the Enzo FS presents the most acceptable balance between the mean relative error and the runtime performance. On the other hand, the Enzo RS is the only heuristic which worked beyond the 4 dimensions.

5. Conclusions

Considering the obtained results from the experiments, we can conclude that the Hilbert space-filling curve can operate in a low-dimensional space (up to thirty dimensions), in terms of index and search. The main conclusions can be summarized as follows:

- The Hilbert curve requires high space resources when generating the cell keys to apply similarity search techniques. Therefore, the curve seems to be more suited to highly dense spaces when used as a direct access;
- The Hilbert curve seems to be more suited to sparse spaces when applied to the original dataset combined with a B-tree variant.

The main contributions of this work are summarized below:

- I tested the performance of the Hilbert space-filling curve as a direct access method with two heuristics up to 36 dimensions. The results indicated that the curve generates too much data making it difficult to use the curve beyond the 4 dimensions.
- I tested the performance of the same curve as a secondary key combined with a B-tree variant up to 36 dimensions. The results showed that this combination worked in the six spaces tested inclusive in the space with 36 dimensions. Compared with a linear search, it is generally faster as the number of dimensions and the order of the curve increases, although it provided approximate results.

Despite the interesting results, that are many aspects on which this work could be improved. I suggest the following:

- Both HZO and Enzo FS did not compute theirs results in spaces bigger than four dimensions due to space restrictions. The experiments could be repeated in another computer with higher space resources in order to try to compute the experiments that did not provide results due to this problem.
- The query group points could be increased in order to test, for example, 100 points instead of the 10 tested in this work. A larger group could provide a more solid set of results.
- It would also be interesting to evaluate the heuristic considering another evaluation metrics as the number of accesses to the disk or memory. It would allow to analyze the clustering property empirically beyond the three dimensions tested in [19].
- The heuristics proposed are simple and had the purpose of testing the Hilbert curve performance as a low-dimensional index. However, a more complete heuristic could be explored in order to optimize the use of the space-filling curves.
- Once the curve worked up to 36 dimensions, it could be tested in higher dimensions applied directly to the original data, as the Enzo RS. It would be interesting to know the Hilbert curve limitations in terms of dimensions.

Acknowledgements

The author would like to thank her advisor Prof. Dr. Andreas Wichert, her family (husband, mother and sister) and closest friends (Ana Silva and Jose Leira). This work is dedicated to her beloved son Enzo.

References

[1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.

[2] J. L. Bentley and J. H. Friedman. Data structures for range searching. *ACM Comput. Surv.*, 11(4):397–409, Dec. 1979.

[3] S. Berchtold, C. Böhm, and H.-P. Kriegal. The pyramid-technique: Towards breaking the curse of dimensionality. *SIGMOD Rec.*, 27(2):142–153, June 1998.

[4] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The x-tree: An index structure for high-dimensional data. pages 28–39, 1996.

[5] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.*, 33(3):322–373, Sept. 2001.

[6] A. R. Butz. Alternative algorithm for hilbert's space-filling curve. *IEEE Transactions on Computers*, 20(4):424–426, 1971.

[7] H.-L. Chen and Y.-I. Chang. All-nearest-neighbors finding based on the hilbert curve. *Expert Syst. Appl.*, 38(6):7462–7475, June 2011.

[8] B. Clarke, E. Fokoue, and H. Zhang. *Principles and Theory for Data Mining and Machine Learning*. Springer Series in Statistics. Springer New York, 2009.

[9] D. Comer. Ubiquitous b-tree. *ACM Comput. Surv.*, 11(2):121–137, June 1979.

[10] C. Faloutsos. Gray codes for partial match and range queries. *IEEE Trans. Softw. Eng.*, 14(10):1381–1393, Oct. 1988.

[11] C. Faloutsos and S. Roseman. Fractals for secondary key retrieval. In *Proceedings of the eighth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '89, pages 247–252, New York, NY, USA, 1989. ACM.

[12] R. Finkel and J. Bentley. Quad trees: a data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[13] T. C. Hales. Jordans proof of the jordan curve cheorem. *Studies in Logic, Grammar and Rhetoric*, 10(23):45–60, 2007.

[14] C. Hamilton. Compact hilbert indices. Technical report, Faculty of Computer Science, 6050 University Ave., Halifax, Nova Scotia, B3H 1W5, Canada, 2006.

[15] C. H. Hamilton and A. Rau-Chaplin. Compact hilbert indices: Space-filling curves for domains with unequal side lengths. *Inf. Process. Lett.*, 105(5):155–163, Feb. 2008.

[16] J. Lawder and P. King. Using space-filling curves for multi-dimensional indexing. In B. Lings and K. Jeffery, editors, *Advances in Databases*, volume 1832 of *Lecture Notes in Computer Science*, pages 20–35. Springer Berlin Heidelberg, 2000.

[17] D. B. Lomet and B. Salzberg. The hb-tree: a multiattribute indexing method with good guaranteed performance. *ACM Trans. Database Syst.*, 15(4):625–658, Dec. 1990.

[18] N. Mamoulis. *Spatial Data Management*. Synthesis Lectures on Data Management. Morgan & Claypool, 2012.

[19] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, 13:2001, 2001.

[20] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst.*, 9(1):38–71, Mar. 1984.

[21] Ooi, K. J. Mcdonell, and S. R. Davis. Spatial kd-tree: An Indexing Mechanism for Spatial Database. *COMPSAC conf.*, pages 433–438, 1987.

[22] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill international editions: Computer science series. McGraw-Hill Companies,Incorporated, 2002.

[23] J. T. Robinson. The kdb-tree: a search structure for large multidimensional dynamic indexes. In *Proceedings of the 1981 ACM SIGMOD international conference on Management of data*, pages 10–18. ACM, 1981.

[24] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[25] A. Wichert. Content-based image retrieval by hierarchical linear subspace method. *J. Intell. Inf. Syst.*, 31(1):85–107, Aug. 2008.

[26] A. Wichert. Subspace tree. In *Content-Based Multimedia Indexing, 2009. CBMI '09. Seventh International Workshop on*, pages 38 –43, june 2009.

[27] A. Wichert. *Intelligent Big Multimedia Databases*. World Scientific, 2015.

[28] C. Yu. *High-Dimensional Indexing: Transformational Approaches to High-Dimensional Range and Similarity Searches*. Lecture notes in computer science. Springer, 2002.