Fractals for Secondary Key Retrieval

Christos Faloutsos†
Shari Roseman

University of Maryland, College Park

ABSTRACT

In this paper we propose the use of fractals and especially the Hilbert curve, in order to design good distance-preserving mappings. Such mappings improve the performance of secondary-key- and spatial- access methods, where multi-dimensional points have to be stored on an 1-dimensional medium (e.g., disk). Good clustering reduces the number of disk accesses on retrieval, improving the response time. Our experiments on range queries and nearest neighbor queries showed that the proposed Hilbert curve achieves better clustering than older methods ("bit-shuffling", or Peano curve), for every situation we tried.

Categories and Subject Descriptors: H.2.2 [**Database Management**]: Physical Design-*access methods*; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing-*indexing methods*
General Terms: Algorithms, Design, Performance
Additional Keywords: distance preserving mappings, geometric data.

1. INTRODUCTION

In this work we propose some space-filling curves which achieve superior distance-preserving mappings. Informally, a space-filling curve is a continuous path which visits every point in a k-dimensional grid exactly once and never crosses itself. The space-filling curves provide a way to order linearly the points of a grid. The goal is to preserve the distance, that is, points which are close in space and represent similar data should be stored close together in the linear order. The space-filling curves are a special case of fractals [10].

Distance preserving mappings are useful in two general situations: a) when we have to manage multi-dimensional points (geometric data) and b) when we have to store a k-dimensional array on the disk. Multidimensional data appear in many applications. In all of them, the performance is improved if the data (=points) are clustered in groups of similar points. The main motivation for this work is efficient secondary key retrieval. A record with k attributes corresponds to a point in a k-d space (see Figure 1.1).

A class of secondary key methods is based on the idea of distance preserving mappings, using "bit-shuffling" or "z-ordering" [14], which is essentially the Peano curve. The binary representations of the attribute values are combined ("shuffled") to create a single value, the "z-value", which can be used as the primary key in conjunction with any primary-key access method. Thus, any primary key access methods gives immediately rise to a secondary key access method. The performance of the latter clearly depends on how good a clustering the distance-

---

† Also with University of Maryland Institute for Advanced Computer Studies (UMIACS).

This research was sponsored partially by the National Science Foundation under the grants DCR-86-16833 and IRI-8719458

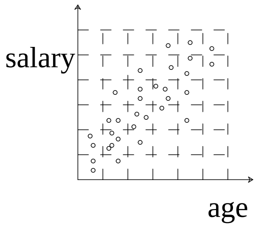


Figure 1.1.
Records of employees correspond to multidimensional points.

preserving mapping achieves.

Moreover, older secondary key methods can benefit from the present work: E.g., the grid file [12] needs to store a k-d directory on the disk; a good way to store it would definitely reduce the number of disk accesses on partial match and range queries.

Additional applications for distance-preserving mappings include the following:

1) Cartography. Maps could be stored and searched electronically, answering efficiently geometric queries [5], [17]. In the TIGER project at the U.S. Bureau of Census, the map of the United States will eventually be stored in a database [19]; the "bit-shuffling" method is used for a distance-preserving mapping.

2) Computer-Aided Design (CAD). For example, VLSI design systems need to store many thousands of rectangles [15] representing electronic gates and higher level elements. Rectangles can be divided in pieces; each piece is assigned a "z-value", according to the Peano curve [13].

3) Computer vision and robotics.

4) Retrieval in large knowledge bases [9], [11], [18].

5) Clustering of data in data base machines [3], [4].

6) In numerical analysis, large k-d arrays that have to be stored on disk [6].

7) In computational geometry. Heuristics in geometric complexity problems use distance-preserving mappings: E.g., to solve the traveling salesman problem, the cities are ordered in a linear ordering, and visited in this order. [1].

The three space-filling curves we compare are the Peano curve, the reflected binary gray-code (RBG) curve [7] and the Hilbert curve. The first two have been used before as distance preserving mappings for partial match retrieval. The hypothesis is that the proposed Hilbert curve yields a better distance preserving mapping, because it avoids long jumps between points. Among the different types of queries [16], we focus on **range queries** and **nearest neighbor queries**; based on them, we derive measures to quantify the "goodness" of a distance preserving mapping.

The structure of the paper is as follows: Section 2 shows how these three space-filling curves are recursively derived. Section 3 briefly describes the experiments on range queries and nearest neighbor queries. Section 4 discusses the results of the experiments. Appendix A lists the algorithms used in the experiments.

2. SURVEY - SPACE-FILLING CURVES

In general, space-filling curves start with a basic path on a k-dimensional square grid of side 2. The path visits every point in the grid exactly once without crossing itself. It has two free ends which may be joined with other paths. The basic curve is said to be of order 1. To derive a curve of order i, each vertex of the basic curve is replaced by the curve of order i−1, which may be appropriately rotated and/or reflected to fit the new curve.

The basic Peano curve for a 2*2 grid, denoted N_1, is shown in Fig. 2.1. To derive higher orders of the Peano curve, replace each vertex of the basic curve with the previous order curve. Fig. 2.1 also shows the Peano curve of order 2 and 3.

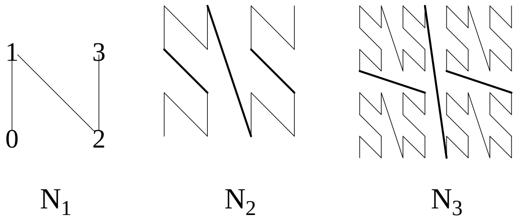


Fig. 2.1 Peano curves of order 1, 2, and 3

The basic reflected binary gray-code curve of a 2*2 grid, denoted R_1 is shown in Fig. 2.2. The procedure to derive higher orders of this curve is to reflect the previous order curve over the x-axis and then over the y-axis. Fig. 2.2 also shows the reflected binary gray-code curve of order 2 and 3.

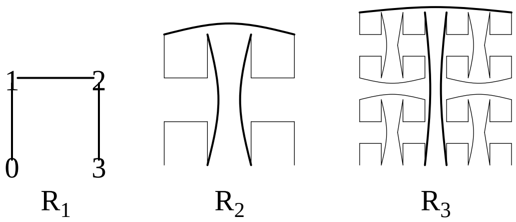


Fig. 2.2 Reflected binary gray-code curves of order 1, 2, and 3

The basic Hilbert curve of a 2*2 grid, denoted H_1, is shown in Fig. 2.3. The procedure to derive higher orders of the Hilbert curve is to rotate and reflect the curve at vertex 0 and at vertex 3. The curve can keep growing recursively by following the same rotation and reflection pattern at each vertex of the basic curve. Fig. 2.3 also shows the Hilbert curves of order 2 and 3. An algorithm to draw this curve is given in Griffiths [8] and Wirth [20].

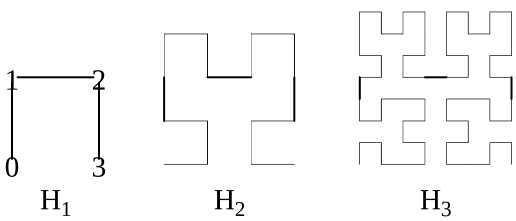


Fig. 2.3 Hilbert curves of order 1, 2, and 3

The path of a space-filling curve imposes a linear ordering, which may be calculated by starting at one end of the curve and following the path to the other end. Orenstein [14] used the term **z-ordering** to refer to the ordering of the Peano curve. He also used the term **z-value** to refer to the order of the point in the z-ordering. We shall use the same terminology here and we shall introduce the terms **h-ordering** and **h-values** to refer to the corresponding ordering and values of the Hilbert curve. Similarly, we use **r-ordering** and **r-values** for the RBG curve. Fig. 2.4 gives an example of the ordering imposed by the Peano curve, the RBG curve, and the Hilbert curve of order 2.

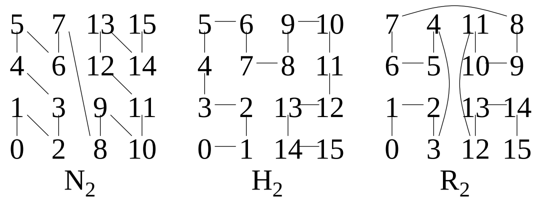


Fig. 2.4 Z-ordering of the Peano curve, r-ordering of the RBG curve, and h-ordering of the Hilbert curve of order 2

Table T2.1 lists the symbols and their definitions

symbol	definition
n	order of the curve
N	size of the side of the grid (= 2^n)
k	number of dimensions

Table 2.1. List of symbols

3. EXPERIMENTS.

Since our goal is to find the best distance preserving mapping, we have to define a measure for the "goodness". In this section we describe two such measures and we justify their use.

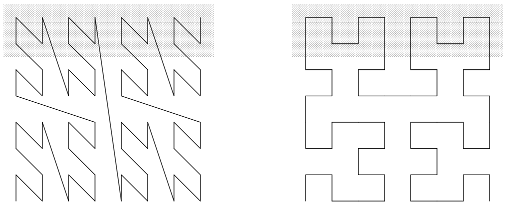
The first measure is related to the performance of a distance preserving mapping under range queries. The setting we have in mind is as follows: We are given a set of multidimensional points; we use a distance-preserving mapping to assign a value (say, "x"-value) to each point; we sort the points according to their "x"-values (where "x" is in the set {r, h, z}) and store them on disk pages. Note that not all possible "x"-values need to be present; only the existing points are stored. In this setting, the best measure for the response time is the number of disk accesses that the average range query requires.

The number of disk accesses depends not only on the mapping, but also on the capacity of the disk pages, the number and distribution of data points etc. A measure that depends only on the mapping is the number of *clusters* that the average query retrieves.

Definition: For a given distance-preserving mapping "x" that maps the points of a k-d grid on 1-d, a **cluster** is defined to be a group of points with consecutive "x"-values.

The proposed measure is a good indication of how good a clustering a distance-preserving mapping can achieve: If a range query retrieves few clusters, then it is likely to require few disk accesses on the actual file.

Figure 3.1 illustrates a range query where the Hilbert curve is better than the Peano curve. The shaded area is the range query. The Peano curve has four pieces (clusters) in the shaded area, while the Hilbert curve only has two.



N_3 H_3

Fig. 3.1 Illustration of clusters for
the Peano curve and the Hilbert Curve

The second proposed measure is related to nearest neighbor queries. An example of a nearest neighbor query is to find the closest gas station from the point of the accident (See Figure 3.2). The entire grid is a map of a neighborhood, which is stored in a database. X represents the point of the accident and G represents the gas stations in the area. The lines represent street blocks and the numbers represent the order we stored the information in the database (using the h-ordering for the Hilbert curve).

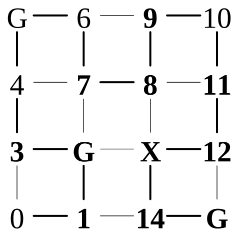


Fig. 3.2 An example of a nearest neighbor query

A straightforward algorithm to solve this problem is as follows:

1. Calculate the h-value of X
2. Find X's preceeding and succeeding points on the Hilbert path. Continue following both ends of the path until one of the points corresponds to a gas station. In our example, G_{15} is the first one found.
3. Calculate the Manhattan distance from G_{15} to X. In our example, this distance d is two blocks.
4. Check all points which are within d(=2) blocks of X (on the two dimensional grid) to see if any closer gas station exists. From Figure 3.2 we see that such a gas station exists which is only one block away from X.

Figure 3.3 is a one-dimensional diagram which shows the points which are within two blocks of X on the two-dimensional grid. The points which are accessed are in the shaded regions of the diagram.

In the above algorithm, it is desirable to have the neighbors which are close on the linearized path indeed close in the k-dimensional space. This gave rise to the second proposed measure, the *maximum neighbor distance*: For a given distance preserving mapping and a given radius

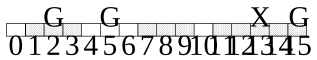


Fig. 3.3 The shaded regions of this one-dimensional diagram illustrate the points which are accessed in step 4 of the above example.

R, the maximum neighbor distance of a point X is the largest Manhattan distance (in the k-d space) of the points within distance R on the linear curve.

To carry out the experiments, we needed algorithms that calculate the "x"-value of a point given its coordinates, as well as the inverse. These algorithms are listed in Appendix A.

4. RESULTS

We experimented with the Hilbert curve, the Peano curve and the RBG curve of two, three and four dimensions.

Tables 4.1 and 4.2 show the average number of clusters for all the possible range queries, for 2 and 3 dimensions, respectively. Table 4.3 shows the average number of clusters for all 3*3*3*3 shaped range queries on the four dimensional grid. Three*three*three*three shaped range queries means each attribute (dimension) has a range of three values. The reasons the queries are limited on the fourth dimension is because of the large number of queries possible. Due to the large number of points in the three- and four-dimensions, only the smaller orders of the curves are tested. The first column of the Tables gives the order n of the curve and the second gives the number of points N (= 2^n) on each side of the grid.

Order n	N*N GRID	HILBERT	RBC	PEANO
1	2*2	1.11	1.11	1.22
2	4*4	1.64	1.92	2.16
3	8*8	2.93	4.02	4.41
4	16*16	5.60	8.71	9.29

Table 4.1 Average number of clusters for all possible range queries for two-dimensional curves.

Order n	N*N*N GRID	HILBERT	RBC	PEANO
1	2*2*2	1.33	1.33	1.59
2	4*4*4	3.72	3.44	4.49

Table 4.2 Average number of clusters for all possible range queries for three-dimensional curves

Note that the number Q of possible range queries is exponential on the dimensionality k of the space:

$$Q=\left[\frac{N(N+1)}{2}\right]^k \qquad (1)$$

Order n	N*N*N*N GRID	HILBERT	RBC	PEANO
1	4*4*4*4	24.74	28.00	40.00
2	8*8*8*8	26.63	29.37	40.33

Table 4.3 Average number of clusters for all 3*3*3*3 shaped range queries for four-dimensional curves

For large values of N or k, the number of queries is too large, making the experimentation difficult.

The results show that the Hilbert curve usually requires fewer clusters than the Peano curve and the reflected binary gray-code curve. The only time the RBG curve does better is for the three dimensional curves of order 2. Both the Hilbert curve and the RBG curve consistently do better than the Peano curve.

Tables 4.4, 4.5 and 4.6 show the average farthest distance from each point to its neighbor points (points whose "x"-values are within N/2 numbers of the current point’s "x"-value), for 2, 3 and 4 dimensions, respectively. Again, the results show that the Hilbert curve generates a better distance-preserving mapping than the other two mappings. In this application, the surprising results are that for large orders of n on a two-dimensional grid, the Peano curve does better than the RBG curve.

Order n	N*N GRID	HILBERT	RBC	PEANO
1	2*2	1.00	1.00	1.50
2	4*4	2.00	2.75	2.75
3	8*8	3.28	5.00	4.84
4	16*16	4.89	8.52	7.91

Table 4.4 Average farthest distance of the neighbors of all the points, for two-dimensional curves.

Order n	N*N*N GRID	HILBERT	RBC	PEANO
1	2*2*2	1.00	1.00	2.00
2	4*4*4	2.00	2.50	3.31
3	8*8*8	3.23	4.04	5.10
4	16*16*16	4.20	5.61	7.03

Table 4.5 Average farthest distance of the neighbors of all the points, for three-dimensional curves.

The contributions of this work are:

- the proposal of the Hilbert curve and, in general, of fractals to achieve good clustering for secondary key retrieval.

Order n	N*N*N*N GRID	HILBERT	RBC	PEANO
1	2*2*2*2	1.00	1.00	2.38
2	4*4*4*4	2.02	2.28	3.50

Table 4.6 Average farthest distance of the neighbors of all the points, for four-dimensional curves.

- the experimentation, that indicates that the proposed Hilbert curve achieves better results than the z-ordering (Peano curve) and the RBG curve for processing range and nearest neighbor queries.

Future research includes:

- Effort to find the "best" distance-preserving mapping.
- Experiments with dimensions greater than four.
- Analytical derivation of the above measures.

APPENDIX A : ALGORITHMS

This appendix lists the algorithms that were used to compute the z-values, the r-values and the h-values of the Peano curve, the RBG curve and the Hilbert curve, respectively. It also lists the algorithms that were used to compute the average number of clusters required for all possible range queries and the average farthest distance of neighbors.

A.1 Computing z-values, r-values and h-values

The z-values, r-values, and h-values are computed by interleaving the bits of the binary representation of coordinates of the point. For the z-values of the two-dimensional Peano curve the algorithm is as follows:

Algorithm P

1. Read in the binary representation of the x and y coordinates.
2. Interleave the bits of the two binary numbers into one string. See Fig. A.1 for an example.

x = 0 0 0 1 y = 0 1 1 0 point (1 , 6)

string = (0 0 0 1 0 1 1 0 $)_2$ = $(22)_{10}$

Fig. A.1 Bit interleaving (Orenstein and Merrett) [14]

3. Calculate the decimal value of the resulting binary string.
   (Note: The the z-value of the point (1,6) in Fig. 4 is 22)

Higher dimensions of the Peano curve are calculated the same way. For k dimensional curves, read and interleave k numbers instead of two numbers.

The algorithm to compute r-values for the RBG is similar to the Peano curve [7]. The only difference is that the bit strings are considered as gray-codes instead of binary codes. Specifically:

Algorithm R

1. Read in the (integer) x and y coordinates.
2. Convert them to the equivalent gray codes (ie., bit strings).
3. Interleave the bits of the two bit strings
4. Consider the resulting bit string as a gray-codeword, and calculate its decimal value.

The generalization for k dimensions and the inverse algorithm are obvious.

Since the Hilbert curve is more complex than the other two curves, we list three different algorithms for calculating the h-values for the Hilbert curve, in increasing order of readability. The first one calculates the h-values for a two-dimensional Hilbert curve. It is the easiest to understand and shows the rotation and reflection explicitly. The second algorithm, which may be used only for three-dimensional curves, is a special case of the third algorithm, which is general and may be used for a Hilbert curve of arbitrary dimensionality.

The algorithm to compute the h-values of the two-dimensional Hilbert curve on a $2^n * 2^n$ grid is:

Algorithm H1

1. Read in the (n-bit) binary representation of the x and y coordinates.
2. Interleave bits of the two binary numbers into one string, i.e., the same way as for the Peano curve.
3. Divide the string from left to right into 2-bit strings, s_i for i=1,...,N.
4. Give a decimal value, d_i, for each two bit string according to the following chart.

   ’00’ equals 0
   ’01’ equals 1
   ’10’ equals 3
   ’11’ equals 2

   and put into an array in the same order as the strings occurred. (This gives the h-values of the basic Hilbert curve.)
5. For each number i in the array, if ...

   i=0 then switch every following occurrence of 1 in the array to 3 and every following occurrence of 3 in the array to 1;

   i=3 then switch every following occurrence of 0 in the array to 2 and every following occurrence of 2 in the array to 0;

   (This makes up for the rotation and reflection of the curves of order higher than 1.)
6. Convert each number in the array to its binary representation (two-bit strings), concatenate all the strings in order from left to right, and calculate the decimal value.

Example 2 Calculate the h-value of the point (1,2) for the Hilbert curve of order 3.

Step 1: x = 0 0 1 y = 0 1 0

Step 2: string = 0 0 0 1 1 0

Step 3: s1 = 00 s2 = 01 s3 = 10

Step 4: d1 = 0 d2 = 1 d3 = 3

Step 5: for d1
d2 = 3 d3 = 1

(these are the only switches)

Step 6: s1 = 00 s2 = 11 s3 = 01
string = 0 0 1 1 0 1
h-value = 13

(If you look at point (1,2) in H_3 in Fig. 2.2, you will notice it is the thirteenth point in the ordering.)

The H1 algorithm has complexity $O(n^2)$ and may only be used for two-dimensional curves. An O(n) algorithm to compute the h-values of the Hilbert curve was derived by Bially [2]. He describes a way to create state transition machines to determine h-values for any dimensionality Hilbert curves. Fig. A.2 shows a state transition machine he created for the three-dimensional Hilbert curve. The states, which are the circles in Fig. A.2, are k*k matrices, which account for the rotation and reflection of the curve. Each state has arrows coming in and out of it. The unbracketed k-tuple on the arrow represents the input and bracketed k-tuple represents the output after the transformations have taken place. In algorithm H2 which calculate the h-values of the Hilbert curve on a $2^n*2^n*2^n$, we use the state diagram created by Bially, which is shown in Fig. A.2.

Algorithm H2

1. Read in the binary representation of the x, y and z coordinates.
2. Interleave bits of the three binary numbers into one string, i.e., the same way as for the Peano curve.
3. Divide the string from left to right into 3-bit strings, but retain the order of the strings.
4. For each three-bit string...

    Change the string according to the output string from the current state and move to the new state pointed to by arrow.

5. Concatenate all the new three bit strings in order and calculate the decimal value.

Example 3 Calculate the h-value of the point (1,2,0) for the Hilbert curve of order 3.

Step 1: x = 0 0 1 y = 0 1 0 z = 0 0 0

Step 2: string = 0 0 0 0 1 0 1 0 0

Step 3: s1 = 000 s2 = 010 s3 = 100

Step 4: current state = 1
for s1: s1 = 000 new state = 9
for s2: s2 = 001 new state = 1
for s3: s3 = 111 new state = 2

Step 5: string = 0 0 0 0 0 1 1 1 1
h-value = 15

Algorithm H2 may only be used with the three-dimensional Hilbert curve, because it specifically uses the state transition machine for the three-dimensional Hilbert curve. The algorithm to calculate the inverse is the same as algorithm H2 with the input and output interchanged in the state diagram.

The following two algorithms create a generic algorithm which may be used with Hilbert curves of any dimension. The first algorithm is done by hand to create some fundamental data which is used in the second algorithm. The fundamental data provides the essential information needed to build a state transition machine of a particular-dimension Hilbert curve. The data, which includes

Fig. A.2 State diagram of the Hilbert curve for three dimensions
(Bially) [2]

information about the basic curve, the rotation and the reflection, will be different for each dimension. To calculate the h-values for points in four-dimensional space, we created our own state diagram using the method described by Bially [2].

The following is the algorithm derived by Bially to calculate the fundamental data. The algorithm here is simplified from the discussion in Bially's paper [2]. The table this algorithm derives is in Table TA.1.

Algorithm H3a : Fundamental Data

1. The first column represents the transformation of the basic rotation and reflection. It is the output from the basic state and is derived by counting in binary.
2. The second column represents the input into the basic state and is derived by counting in any Gray code. (Table TA.1 uses the reflected binary code.)
3. For each entry in the second column, the third column will have two entries. The very first entry in the table must be 0000 and the very last entry in the table must be the last entry in the second column. (In Table TA.1, the last entry of the third column must be

1000.) The rest of the entries must follow these rules:

1. Each pair of adjacent entries in the third column must differ in one bit position exactly. (i.e., ...,1100,1000,1010,1000,...)
2. For each pair of adjacent entries in the second column, say A and B, the second entry in the third column corresponding to entry A and the first entry in the third column corresponding to entry B must differ in exactly the same bit position that entries A and B differ.

4. The fourth column is derived by placing a '1' in the only bit position which changes between the pair of adjacent entries in the third column which have the same corresponding entry in the second column. Place a '0' in all other bit positions.
5. The last column is derived by forming a P matrix which satisfies
(P)*(entry in fourth column) = last entry in second column
(i.e., in this table, the last entry is 1000) P must be formed so there is exactly one '1' in each column and row.
6. The positive and negative ones of matrix P in the fifth column are determined by the first entry of the pair of entries in the third column which have the same corresponding entry in second column. If the i-th element of this first entry is a '1' then the element in the i-th column of the matrix P is '-1', otherwise it is positive one.

The hardest part of this algorithm is step 3. While the rules may look easy, it is hard to implement so that all the rules are satisfied simultaneously. Also, there may be more than one derivation of step 3.

Once the fundamental data is established, a computer program may calculate all the other information for the state diagram as it is needed in the calculation for the h-value. This is also described in Bially's paper. The algorithm to compute the h-values of the four dimensional Hilbert curve on a $2^n*2^n*2^n*2^n$ grid is:

Algorithm H3b : General Algorithm for the Hilbert Curve

0. Initialize the fundamental data from Algorithm H3a. Any state in the Table TA.1 may be used as the initial state to produce a space filling curve. In this experiment, we used the first matrix, P, in table TA.1.
1. Read in the binary representation of the x, y, z, and t coordinates.
2. Interleave bits of the four binary numbers into one string, i.e., the same way as for the Peano curve.
3. Divide the string from left to right into 4-bit strings, input, but retain the order of the strings.
4. For each four bit string...
    1. For j = 0 to 3
        1. Find the nonzero element in the j-th row of the current matrix.
        2. Set i = the column which corresponds to that nonzero element.
        3. If the nonzero element = 1 then output[j] = input[i]
        else if the nonzero element = -1 then output[j] = 1 - input[i]
    2. Using the fundamental data (Table TA.1), search the second column for the calculated four bit output string. The corresponding entry in the first

Output	Input			P
0000	0000	0000 0001	0001	0 0 0 1 0 0 1 0 0 1 0 0 1 0 0 0
0001	0001	0000 0010	0010	0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 0
0010	0011	0000 0100	0100	0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0
0011	0010	0101 1101	1000	1 0 0 0 0 -1 0 0 0 0 1 0 0 0 0 -1
0100	0110	1001 1011	0010	0 0 1 0 0 0 0 -1 0 1 0 0 -1 0 0 0
0101	0111	1010 1000	0010	0 0 -1 0 0 0 0 1 0 1 0 0 -1 0 0 0
0110	0101	1010 1000	0010	0 0 1 0 0 0 0 1 0 1 0 0 -1 0 0 0
0111	0100	1001 1011	0010	0 0 1 0 0 0 0 -1 0 1 0 0 -1 0 0 0
1000	1100	0011 1011	1000	1 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 -1
1001	1101	1010 0010	1000	-1 0 0 0 0 1 0 0 0 0 -1 0 0 0 0 1

Table TA.1 (Table continued on next page)

column is the new four bit rotated string.

3. Calculate the new current state...

1. Retrieve the matrix P_{basic} from Table TA.1 which corresponds to that entry in the first column which is the rotated string.

Output	Input			P
1010	1111	0000 1000	1000	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1
1011	1110	1001 1011	0010	0 0 1 0 0 0 0 -1 0 1 0 0 -1 0 0 0
1100	1010	1111 1101	0010	0 0 -1 0 0 0 0 -1 0 -1 0 0 -1 0 0 0
1101	1011	1100 1000	0100	0 -1 0 0 0 0 0 1 0 0 1 0 -1 0 0 0
1110	1001	1010 1000	0010	0 0 -1 0 0 0 0 1 0 1 0 0 -1 0 0 0
1111	1000	1001 1000	0001	0 0 0 -1 0 0 1 0 0 1 0 0 -1 0 0 0

Table TA.1 Calculations of fundamental data
for Hilbert curve of four dimensions.

2. Multiply that state by the current state using matrix multiplication. $P_{new} = P_{basic} * P_{current}$

5. Concatenate all the four bit rotated strings in order and calculate the decimal value.

Note that the fundamental data algorithm and the generic algorithm need only be modified slightly to handle n-dimensional Hilbert curve. Wherever there are four-bit strings, there will be n-bit strings. The algorithm to determine the inverse will be the same as algorithm H3.b except columns 1 and 2 of table TA.1 will be interchanged.

Example 4 Calculate the h-value of the point (2,1,3,0)for the Hilbert curve of order 2.

Step 0: current state =

0 0 0 1
0 0 1 0
0 1 0 0
1 0 0 0

Step 1: x_1 = 1 0 x_2 = 0 1 x_3 = 1 1 x_4 = 0 0

Step 2: string = 1 0 1 0 0 1 1 0

Step 3: s1: input[0]=1 s2: input[0]=0
input[1]=0 input[1]=1
input[2]=1 input[2]=1
input[3]=0 input[3]=0

Step 4: for s1

4.1) output[0]=input[3]=0
output[1]=input[2]=1
output[2]=input[1]=0
output[3]=input[0]=1

4.2) output = 0 1 0 1 --> 0 1 1 0 = rotated = r_1

4.3) new current state =

$$\begin{matrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{matrix}$$

for s2

4.1) output[0]=input[1]=1
output[1]=input[0]=0
output[2]=input[2]=1
output[3]=input[3]=1

4.2) output = 1 0 1 1 --> 1 1 0 1 = rotated = r_2

4.3) new current state =

$$\begin{matrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \end{matrix}$$

Step 5: string = $r_1 \| r_2$ = 0 1 1 0 1 1 0 1
h-value = 109

A.2. Computing the average number of clusters

The algorithm to compute the average number of clusters of all possible range queries for the Peano curve (of any dimensionality) is

1. Read in the order n of the curve.
2. For each possible range query...

   For each point inside the query range...

   1. Calculate the "x"-value of the point.

2. Calculate the coordinates of the "x"-successor, which is the point whose "x"-value is one higher than the current point.
3. Check if the coordinates of the "x"-successor are within the query range - if not, add one more to the number of clusters.

3. Divide the number of clusters by the number of possible range queries.

The same algorithm may be used to compute the average number of clusters of all possible range queries for the RBG curve and the Hilbert curve. One shortcut may be taken for the Hilbert curve. Only the borderline points of the range query need to go through step 2 of the above algorithm, since all the points inside the query have successor points either inside the query or on the border. The Peano curve and the RBG curve, on the other hand, may have points inside the boundary of the range query which take long diagonal jumps outside the range to get to their successor points.

A.3. Computing the average maximum neighbor distance

The algorithm to compute the average farthest distance of the neighbors (points with close "x"-values) for all the points in the two-dimensional curve is

1. Read in N of a N*N grid, i.e., for the basic curve, N=2. Read in the radius R (all "x"-neighbors within distance R on the "x"-ordering will be considered)
2. For each point ("x"-value) in the curve...
   1. Calculate the coordinates of the current point, $x_{current}$ and $y_{current}$.
   2. For each neighboring point P (which is a point whose "x"-value is within R steps from the current "x"-value)...
      1. Calculate the coordinates of the neighbor point, x_P and y_P.
      2. Calculate the distance the neighbor point is from the current point, using the Manhattan distance. The equation is $| x_{current} - x_P | + | y_{current} - y_P |$
      3. Compare the distance to previous neighbor's distances and find the largest distance.
   3. Add the largest distance calculated in step 2.2.3 step with the other largest distances previously calculated for other points.
3. Divide the sum of the largest distances (the answer from 2.3) by the number of points in the curve.

Note that this algorithm may be easily modified to handle curves of any dimensionality.

References

1. Bartholdi, J.J. and L.K. Platzman, *Heuristics Based on Spacefilling Curves for Combinatorial Problems in the Plane*, 1986. unpublished manuscript
2. Bially, T., ‘‘Space-Filling Curves: Their Generation and Their Application to Bandwidth Reduction,’’ *IEEE Trans. on Information Theory*, vol. IT-15, no. 6, pp. 658-664, Nov. 1969.
3. Boral, H. and S. Redfield, ‘‘Database Machine Morphology,’’ *Proc. 11th International Conference on VLDB*, pp. 59-71, Stockholm, Sweden, Aug. 1985.

4. Cheiney, J.P., P. Faudemay, Rodolphe Michel, and J.M. Thevenin, ''A Reliable Parallel Backend Using Multiattribute Clustering and Select-Join Operations,'' *Proc. 12th International Conference on VLDB*, pp. 220-227, Kyoto, Japan, Aug. 1986.
5. Chock, M., A.F. Cardenas, and A. Klinger, ''Database Structure and Manipulation Capabilities of a Picture Database Management System (PICDMS),'' *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 4, pp. 484-492, July 1984.
6. Duff, I.S., ''Design Features of a Frontal Code for Solving sparse Unsymmetric Linear Systems Out-of-core,'' *SIAM J. Sci. Stat. Computing*, vol. 5, no. 2, pp. 270-280, June 1984.
7. Faloutsos, C., ''Gray Codes for Partial Match and Range Queries,'' *IEEE Trans. on Software Engineering*, vol. 14, no. 10, pp. 1381-1393, Oct. 1988. early version available as UMIACS-TR-87-4, also CS-TR-1796
8. Griffiths, J.G., ''An Algorithm for Displaying a Class of Space-filling Curves,'' *Software-Practice and Experience*, vol. 16, no. 5, pp. 403-411, May 1986.
9. Kowalski, R., ''Logic for Data Description,'' in *Logic and Data Bases*, ed. J. Minker, pp. 77-103, Plenum Press, 1978.
10. Mandelbrot, B., *Fractal Geometry of Nature*, W.H. Freeman, New York, 1977.
11. Minker, J., ''An Experimental Relational Data Base System Based on Logic,'' in *Logic and Data Bases*, ed. J. Minker, Plenum Press, 1978.
12. Nievergelt, J., H. Hinterberger, and K.C. Sevcik, ''The Grid File: An Adaptable, Symmetric Multikey File Structure,'' *ACM TODS*, vol. 9, no. 1, pp. 38-71, March 1984.
13. Orenstein, J., ''Spatial Query Processing in an Object-Oriented Database System,'' *Proc. ACM SIGMOD*, pp. 326-336, Washington D.C., May 1986.
14. Orenstein, J.A. and T.H. Merrett, ''A Class of Data Structures for Associative Searching,'' *Proc. of SIGACT-SIGMOD*, pp. 181-190, Waterloo, Ontario, Canada, April 2-4, 1984.
15. Ousterhout, J. K., G. T. Hamachi, R. N. Mayo, W. S. Scott, and G. S. Taylor, ''Magic: A VLSI Layout System,'' *21st Design Automation Conference*, pp. 152 - 159, Alburquerque, NM, June 1984.
16. Rivest, R.L., ''Partial Match Retrieval Algorithms,'' *SIAM J. Comput*, vol. 5, no. 1, pp. 19-50, March 1976.
17. Roussopoulos, N. and D. Leifker, ''Direct Spatial Search on Pictorial Databases Using Packed R-Trees,'' *Proc. ACM SIGMOD*, Austin, Texas, May 1985.
18. Thom, J.A., K. Ramamohanarao, and L. Naish, ''A Superjoin Algorithm for Deductive Databases,'' *Proc. 12th International Conference on VLDB*, pp. 189-196, Kyoto, Japan, Aug. 1986.
19. White, M., *N-trees: Large ordered Indexes for Multi-dimensional Space*, Application Mathematics Research Staff, Statistical Research Division, U.S. Bureau of the Census, Dec. 1981.
20. Wirth, N., *Algorithms and Data Structures*, Prentice-Hall Inc, Englewood Cliff, NJ, 1986.