

All-Nearest-Neighbors Finding Based on the Hilbert Curve¹

Hue-Ling Chen[†] and Ye-In Chang

[†]Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan
Republic of China
{E-mail: chen.hueling@gmail.com}
{Tel: 886-3-4244395}
{Fax: 886-3-4245281}

Abstract

An all-nearest-neighbors (ANN) query retrieves all nearest neighbors to all query objects. We may perform large number of one-nearest-neighbor queries to answer such an ANN query. Due to no total ordering of spatial proximity among spatial objects, the Hilbert curve approach has proposed to preserve the spatial locality. Chen and Chang have proposed a neighbor finding strategy (denoted as the CCSF strategy) based on the Hilbert curve to compute the absolute location of the neighboring blocks. However, it costs much time during the transformation between the Hilbert curve and the Peano curve. On the other hand, in the strategy based on R or R^* -trees for an ANN query, large number of unnecessary distance comparisons have to be done due to the problem of overlaps within the R -tree, resulting in many redundant disk accesses. Therefore, in this paper, we first propose the one-nearest-neighbor finding strategy directly based on the Hilbert curve (denoted as the ONHC strategy) for a one-nearest-neighbor query. By relations among orientations, orders, and quaternary numbers, we compute the relative locations of the query block and the neighboring block in the Hilbert curve. Then, the nearest neighbor of one query point can be found directly from these neighboring blocks. Next, by using our ONHC strategy, we propose the all-nearest-neighbors finding strategy based on the Hilbert curve (denoted as the ANHC strategy) for an ANN query. Finally, from the simulation result, we show that our ONHC strategy needs less response time (the CPU-time and the I/O time) than the CCSF strategy for the one-nearest-neighbor query. We also show that our ANHC strategy needs less response time than the strategy based on R^* -trees for an ANN query.

(Key Words: Hilbert curve, nearest neighbor queries, R -tree, space filling curves, spatial index)

¹This research was supported in part by the National Science Council of Republic of China under Grant No. NSC-93-2213-E-110-027 and National Sun Yat-Sen University. The authors also like to thank “Aim for Top University Plan” project of NSYSU and Ministry of Education, Taiwan, for partially supporting the research.

1 Introduction

Given two datasets A and B which contain spatial objects, an *all-nearest-neighbors* (ANN) query retrieves for each object in dataset A its nearest neighbor in dataset B . An ANN query is often applied in applications including the sensor networks and urban planning in the geographic information systems (GIS), the spatial data analysis and mining in the spatial databases, and image processing in the multimedia databases [1, 15, 30, 31]. For example, “to find the nearest parking lot or bus station for each high speed rail station” is a common ANN query in the urban planning.

An ANN query can be considered as a hybrid spatial query of spatial joins and the nearest neighbor queries [28, 31]. Take Figure 1 as an example. The result of the spatial join on regions A and B in Figures 1-(a) and (b), respectively, is shown as region C in Figure 1-(c). Region C contains two objects a_3 and b_2 which could be the nearest neighbor to each other. However, the nearest neighbor of object a_3 is object b_1 which is not in region C . Thus, the nearest neighbor cannot be completely obtained by the spatial join. The ANN query should consider not only the spatial predicate “the intersection” considered in the spatial join, but the spatial predicate “the nearest neighbor” on pairs of spatial objects.

In addition, an ANN query between datasets A and B can be considered as large number of the nearest neighbor query (denoted as the ONN query) for each spatial object in dataset A where the nearest neighbor is found in dataset B , and vice versa. An ANN query can be answered based on the neighbor finding for the ONN query. That is, when the nearest neighbor could be efficiently found for the ONN query, the number of disk accesses which results in the response time for the ANN query could be reduced. However, the difficulty of the neighbor finding for the ONN query in the two-dimensional space is no total ordering for spatial objects. In other words, it is difficult to preserve the spatial proximity between the spatial objects. Many *spatial data access methods* [24] for the ONN query have been proposed to provide access paths to the spatial objects. They focused on candidate neighbors only and prune the objects which are too far from the query object. In general, they used the spatial index as an accelerator and can be classified into two categories: the *space-based* strategies and the *object-based* strategies [24]. The space-based strategies have been designed based on the decomposition of the space, *e.g.*, the strategy based on

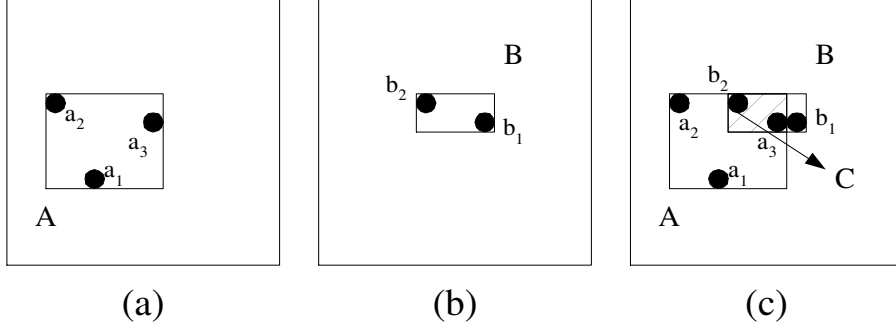


Figure 1: An example of the spatial join: (a) dataset A ; (b) dataset B ; (3) the result in the region C .

the quadtree [10, 27]. They performed some rules on the subdivided space to find the nearest neighbor, including the recursive tree-based and complicated bits-based rules. On the other hand, the object-based strategies have been designed based on the distribution of data objects, *e.g.*, the strategy based on the R -tree or R^* -tree [3, 7, 12, 22, 23, 31]. They required to traverse more than one path in the tree structure and accessed many unnecessary partitions to find the nearest neighbor.

In order to preserve the spatial proximity, a space-filling curve has been proposed as one total ordering which is a mapping between coordinates of spatial objects in the two-dimensional space and sequence numbers in the one dimensional space [5, 6, 11, 13, 16, 20]. Then, spatial objects are stored in the disk according to sequence numbers. Some examples of the space-filling curves are the Peano curve, the RBG curve and the Hilbert curve. The Hilbert curve has the best *clustering* property among all curves, even better than the quadtree [10] and the R -tree [20, 31]. The clustering means that spatial objects which are close to each other in the two-dimensional space could be stored in adjacent blocks of the disk. The number of disk accesses for the neighboring objects could be reduced by sequentially accessing adjacent blocks. The Hilbert curve has been widely used in a variety of fields including spatial databases [5, 14, 16, 20, 29], geographic information systems [11], image processing and compression [8, 17], genome visualization [9] and scientific computing [4, 6, 19]. Some neighbor finding strategies have been proposed based on the Hilbert curve [2, 15, 18]. They performed the range query in the disk to search for the neighboring blocks next to the query block in the two dimensional space. However, it may cost much time

to search for those neighboring blocks which may not stored in the adjacent blocks of the disk. In Chen and Chang’s strategy [5] (denoted as the CCSF strategy), they computed the absolute locations of neighboring blocks by binary bits of the coordinates. Then, the neighboring blocks are directly accessed, whether they are in one or two-dimensional space. However, it costs much time during the transformation between the Hilbert curve and the Peano curve in the CCSF strategy, as shown in Figure 2-(a).

Therefore, in this paper, we propose the one-nearest-neighbor finding strategy directly based on the Hilbert curve (denoted as the ONHC strategy), as shown in Figure 2-(b). We assume that spatial objects are grouped into blocks based on the Hilbert curve and focus on the neighboring blocks finding. We find the relations among the orientation, the order, and the quaternary number of a block in the Hilbert curve. By these relations, we first generate the direction sequence to store the orientation of the query block Q in the Hilbert curve of each order. Next, by direction sequences generation for block Q , we obtain the relative location of the block Q and block NN in the curve of each order. Block NN is the neighboring block of block Q in one of eight directions. We compute the quaternary number of block NN and obtain its sequence number by transformation of the quaternary number from base four to ten. Then, we can sequentially and directly access these neighboring blocks in the disk by their sequence numbers after sorting. Finally, we obtain the nearest neighbor by distance comparisons in these neighboring blocks. As compared to the CCSF strategy shown [5] in Figure 2-(a), our ONHC strategy needs less response time (the CPU time and the I/O time) than the CCSF strategy for the ONN query. Since an ANN query contains large number of the ONN queries, we also propose the all-nearest-neighbors finding strategy based on the Hilbert curve (denoted as the ANHC strategy) by using our ONHC strategy. As compared to the all-nearest neighbor finding strategy based on the R^* -tree (the variation of the R -tree) [7, 31], our ANHC strategy accesses the neighboring blocks sequentially and directly by sorted sequence numbers in the Hilbert curve. Our ANHC strategy does not access unnecessary and duplicated partitions like the strategy based on the R^* -tree. Therefore, our ANHC strategy needs less response time than the strategy based on the R^* -tree.

The rest of this paper is organized as follows. In Section 2, we briefly describe two

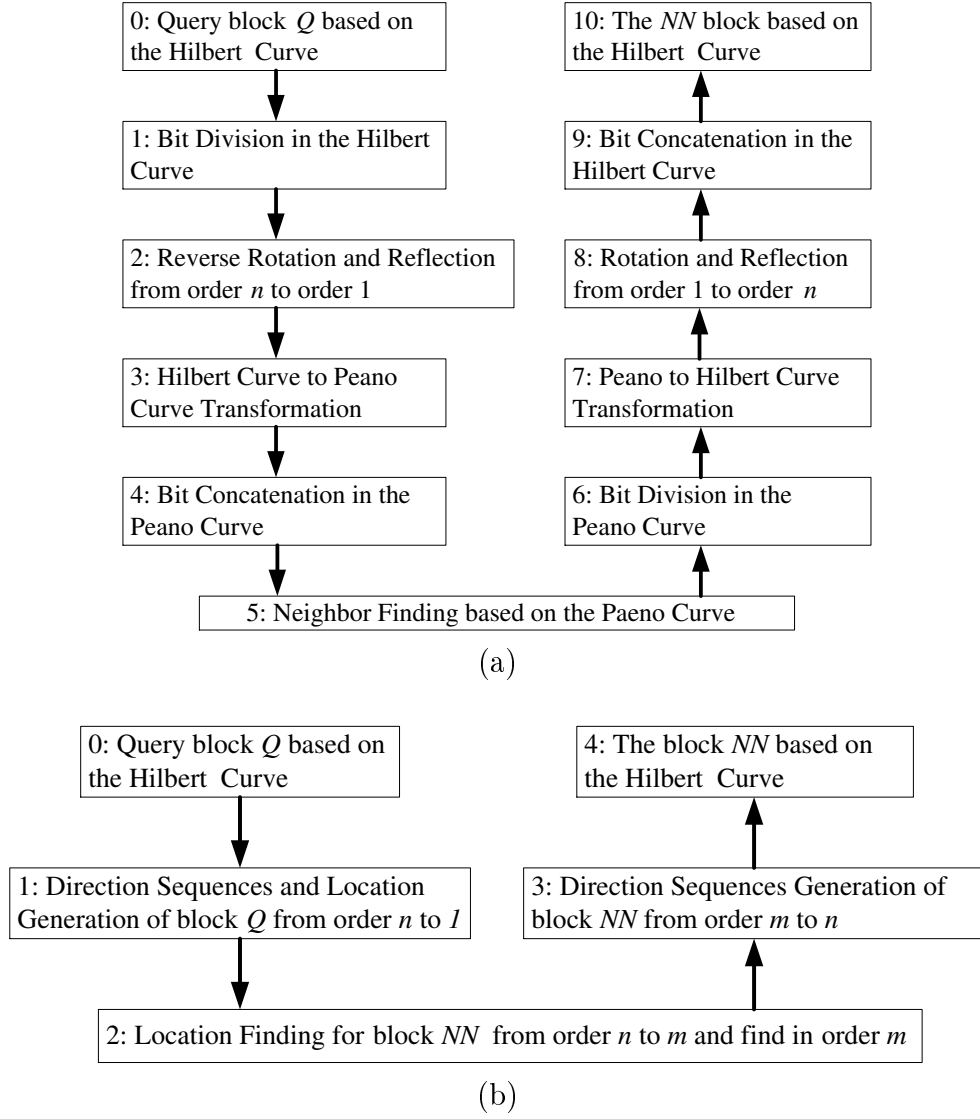


Figure 2: The neighbor finding based on the Hilbert curve: (a) the CCSF strategy; (b) our ONHC strategy.

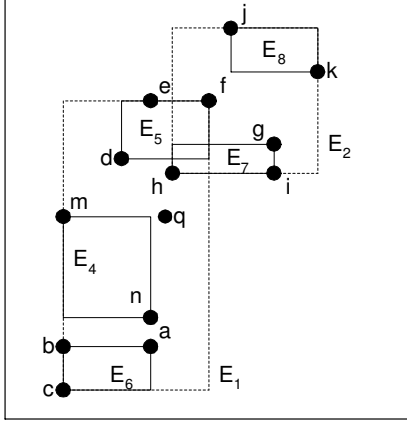
neighbor finding strategies based on the R -tree and the Hilbert curve, respectively. In Section 3, we present the ONHC strategy and the ANHC strategy based on the Hilbert curve. In Section 4, we compare our ONHC strategy with the CCSF strategy in the performance of the ONN query. We also compare our ANHC strategy with the strategy based on the R^* -tree (the variation of the R -tree) in the performance of the ANN query. Finally, we give the conclusion.

2 Related Work

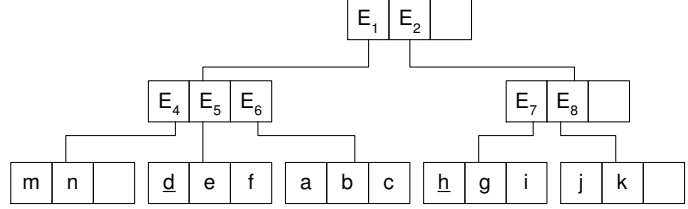
The neighbor finding can work efficiently by the strategy based on the spatial index. The spatial index is used to well organize the spatial data and preserve the spatial proximity. In this section, we briefly introduce two of well-known strategies for the one-nearest-neighbor query based on two spatial indices, including the R -tree [12] and the Hilbert curve [5].

2.1 The R -tree

Existing strategies for the nearest neighbor query assume that the dataset is indexed by an R -tree and use various metrics on $mindist(q, M)$ to prune the search space. The metric $mindist(q, M)$ denotes the minimum distance between the query point q and any point in a minimum bounding rectangle (MBR) M [12, 22, 23, 24, 31]. Take Figure 3-(a) as an example. The entries in Figure 3-(b), including intermediate and leaf entries, are sorted by their $mindist$ from the query point q . The neighbor finding process starts from the root. Since MBR E_1 has the minimum $mindist$, the node of MBR E_1 in the R -tree is visited first. Next, the node of MBR E_4 in that of MBR E_1 is visited. The first candidate point m is retrieved from the leaf node of MBR E_4 . Then, the process should backtrack to the previous level. Although the $mindist$ of MBR E_5 is larger than the one of MBR E_4 , the node of MBR E_5 has to be visited before backtracking again to the root level. The candidate point d in MBR E_5 replaces point m in MBR E_4 , since the $mindist$ of point d is smaller than the $mindist$ of point m . However, point d is the local optimal nearest neighbor. The reason is that there exists an overlap between two MBR's, E_1 and E_2 , respectively. It needs to search in another internal node of MBR E_2 which is overlapped with MBR E_1 . Finally, after searching all paths in the R -tree, the global optimal nearest



(a)



(b)

Figure 3: An example of the R-tree: (a) points and MBR's; (b) the structure of R-tree.

point h will be obtained in some node of MBR E_7 . Consequently, it has to travel more than one tree path and access many partitions of nodes to find the nearest neighbor to the query point in the R -tree [12], and even in the R^* -tree [3] which is a variation of the R -tree to reduce overlaps.

2.2 The Hilbert Curve

Given a two-dimensional square space of size $N \times N$, where $N = 2^n$ with order $n \geq 0$, the Hilbert curve recursively divides the space into four equal-sized blocks. Each block is given a sequence number which ranges from 0 to $N^2 - 1$. For example, Figure 4-(a) shows the Hilbert curve of order $n = 1$ which linearly orders four blocks by four sequence numbers ranged from 0 to 3. The total ordering of the Hilbert curve is that the adjacent blocks in the two-dimensional space always correspond to the adjacent line intervals in the curve [6, 19]. Figure 4-(b) shows the Hilbert curve of order $n = 2$ in which the sequence numbers range from 0 to 15 $(= (2^2)^2 - 1)$. It is derived from the curve of order 1 in Figure 4-(a) with the reflection and rotation on the first and last blocks of the curve of the previous order 1. Then, the orientation (*i.e.*, the traversal path) of these two blocks are changed to preserve the spatial proximity of two adjacent blocks everywhere. Figure 4-(c) shows the Hilbert curve of order $n = 3$ in which sequence numbers range from 0 to 63 $(= (2^2)^3 - 1)$. It is derived

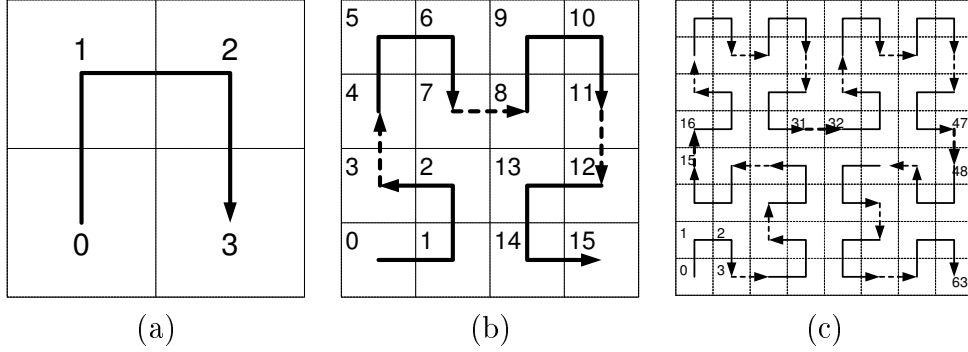


Figure 4: The Hilbert curve in two dimensional space: (a) order $n = 1$; (b) order $n = 2$; (c) order $n = 3$.

from the curve of the previous order 2 after the similar reflection and rotation. Therefore, the Hilbert curve has the better clustering property than the other curves, *e.g.*, the Peano curve [11, 13, 14, 16, 20]. Chen and Chang proposed the neighbor finding strategy based on the Hilbert curve (denoted as the CCSF strategy [5]), where the process is shown in Figure 2. The CCSF strategy uses the bit shuffling property of the Peano curve on the coordinate system and transformation rules between curves in the neighbor finding process.

3 The Neighbor Finding Based on the Hilbert Curve

Because the Hilbert curve has the good clustering property of preserving spatial proximity, we propose the one-nearest-neighbor finding strategy and the all-nearest-neighbors finding strategy based on the Hilbert curve. In Section 3.1, we describe the relations among the orders, the orientations, and the quaternary numbers in the Hilbert curve. These relations will be used in the process of the neighbor finding. In Section 3.2, we present the ONHC strategy for the one-nearest-neighbor query. In Section 3.3, we present the ANHC strategy by using our ONHC strategy for the all-nearest-neighbors query.

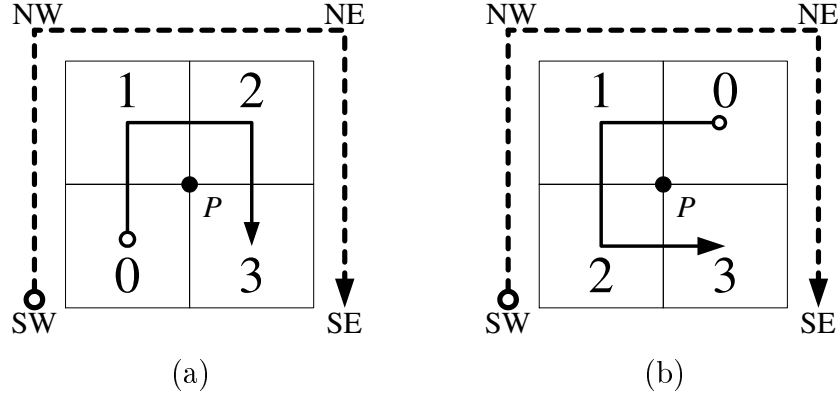


Figure 5: The oriented numbers in the SW, NW, NE, and SE blocks of the center point P : (a) 0, 1, 2, 3; (b) 2, 1, 0, 3.

3.1 Relations

3.1.1 Orientations and Direction Sequences

A Hilbert curve recursively divides the space into four equal-sized blocks due to the limit of the block capacity. The Hilbert curve never maintains the same direction for more than three consecutive blocks. Take the quaternary space in Figure 5 as an example. The quaternary space is divided into four equal-sized blocks such that four blocks are in the southwest (SW), northwest (NW), northeast (NE) and southeast (SE) directions of the center point P . We call these blocks as the SW, NW, NE, and SE blocks. The orientation is defined as the traversal path along the Hilbert curve for four consecutive blocks with four oriented numbers 0, 1, 2, and 3, respectively. For example, the orientation of Figure 5-(a) or (b) is shown as the thick line.

However, in our strategy, we denote oriented number $Orient$ which is the value in base 4 as the sequence number of the block in the orientation. We define the direction sequence $DSQ = (Orient_{SW}, Orient_{NW}, Orient_{NE}, Orient_{SE})$, where $Orient_{SW}$, $Orient_{NW}$, $Orient_{NE}$ and $Orient_{SE}$ are denoted as the oriented numbers of the SW, NW, NE, and SE blocks, respectively, in the quaternary space. For example, direction sequence DSQ of Figure 5-(a) is (0,1,2,3) which sequentially follows the dotted line. By following the same dotted line, direction sequence DSQ of Figure 5-(b) is (2,1,0,3).

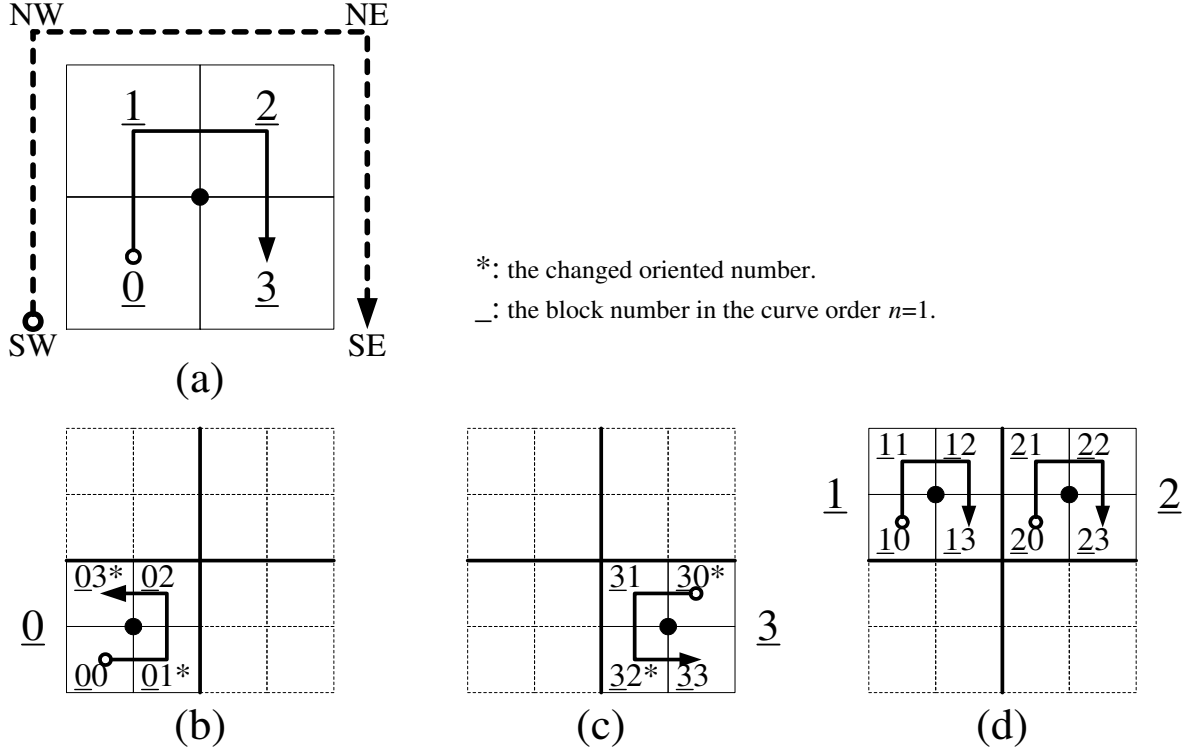


Figure 6: Blocks in the Hilbert curve of order 1 and 2 : (a) four blocks in the curve of order 1; (b) four sub-blocks of block $(\underline{0})_4$ in the curve of order 2; (c) four sub-blocks of block $(\underline{3})_4$ in the curve of order 2; (d) four sub-blocks of blocks $(\underline{1})_4$ and $(\underline{2})_4$ in the curve of order 2.

3.1.2 Quaternary Numbers, Orders, and Direction Sequences

In order to derive the Hilbert curve of the large order (*i.e.*, the order $n > 1$), four blocks in the Hilbert curve of the order n are replicated by the previous curve of the order $(n - 1)$ after the reflection and rotation [5]. Take Figure 6 as an example. While replicating by the Hilbert curve of order 1 for block $(\underline{0})_4$ in Figure 6-(a), the orientation is reflected horizontally, rotated clockwise 90° , as shown in Figure 6-(b). The SW, NW, SE, and NE sub-blocks in block $(\underline{0})_4$ are linearly ordered by the Hilbert curve of order 2. The oriented numbers of these four sub-blocks are 0, 3^* , 2, and 1^* , respectively, which are not underlined in Figure 6-(b). The symbol ‘*’ indicates the changed oriented number in Figure 6-(b) after the reflection and rotation of the orientation in Figure 6-(a).

We concatenate each oriented number $(d_i)_4$ in the Hilbert curve of order i to obtain the quaternary number $(d_1 d_2 \dots d_i \dots d_n)_4$ of the block in the Hilbert curve of order n , where

d_i in $\{0, 1, 2, 3\}$. The quaternary number has n oriented numbers which is related with the order n . Take Figure 6-(b) as an example. Since block $(\underline{0})_4$ is in the Hilbert curve of order 1, four quaternary numbers for its four sub-blocks, the SW, NW, NE, and SE blocks $(\underline{00})_4$, $(\underline{03})_4$, $(\underline{02})_4$ and $(\underline{01})_4$, respectively. These four sub-blocks are ordered by the Hilbert curve of order 2.

We use direction sequence DSQ_{bd} to store the orientation of the block b where the sub-block bd locates in the curve of order n . The notation b is the quaternary number $(d_1 d_2 \dots d_i \dots d_{n-1})_4$ in the curve of order $(n-1)$. The notation d is the oriented number in the orientation of block b . For example, direction sequence DSQ_0 of block $(\underline{0})_4$ in Figure 6-(a) is $(0, 1, 2, 3)$. Direction sequence DSQ_{01} of sub-block $(\underline{01})_4$ in Figure 6-(b) is $(0, 3^*, 2, 1^*)$ which can be derived from direction sequence DSQ_0 after interchanging oriented numbers 1 and 3. Note that the symbol ‘*’ indicates the changed oriented number. In the same way, direction sequences DSQ_{00} , DSQ_{02} and DSQ_{03} can be obtained as $(0, 3^*, 2, 1^*)$.

For the orientation of block $(\underline{3})_4$ in Figure 6-(c), it is obtained after the reflection horizontal and the rotation anti-clockwise 90° of the orientation in Figure 6-(a). The quaternary numbers for SW, NW, NE, and SE sub-blocks, respectively, in block $(\underline{3})_4$ are $(\underline{32})_4$, $(\underline{31})_4$, $(\underline{30})_4$, and $(\underline{33})_4$, respectively. Since direction sequence DSQ_3 is $(0, 1, 2, 3)$, direction sequence DSQ_{30} of sub-block $(\underline{30})_4$ is $(2^*, 1, 0^*, 3)$ after interchanging oriented numbers 0 and 2 in direction sequence DSQ_3 . In the same way, direction sequences DSQ_{31} , DSQ_{32} and DSQ_{33} can be obtained as $(2^*, 1, 0, 3^*)$.

For the orientation of block $(\underline{1})_4$ or $(\underline{2})_4$ in Figure 6-(d), it is the same as the orientation in Figure 6-(a) without the reflection and rotation. Figure 6-(d) shows the quaternary numbers for SW, NW, NE, and SE sub-blocks in blocks $(\underline{1})_4$ or $(\underline{2})_4$. The direction sequences of four sub-blocks in blocks $(\underline{1})_4$ or $(\underline{2})_4$ are the same as direction sequence $DSQ_1 = (0, 1, 2, 3)$ or $DSQ_2 = (0, 1, 2, 3)$. Moreover, we can transform the quaternary number $(d_1 d_2 \dots d_i \dots d_n)_4$ into the decimal value of the sequence number in the curve of order n . For example, quaternary number $(\underline{32})_4$ in Figure 6-(c) can be transformed into sequence number 14 $(= 4^1 * 3 + 4^0 * 2)$ in the curve of order 2 as shown in Figure 4-(b).

Therefore, we assume that the block $(B)_4$ is the sub-block of block $(A)_4$. Blocks $(A)_4$ and $(B)_4$ are ordered by the Hilbert curve of order a and b , respectively, where $a > 0$,

$b - a = 1$. That is, block numbers $(A)_4$ and $(B)_4$ are quaternary numbers which combine the number of a and b oriented numbers, respectively. We conclude the following three cases to derive the direction sequence DSQ_B from DSQ_A depending on the a' th oriented number of block number $(A)_4$.

Case C13. If the a' th oriented number in block number $(A)_4$ is 0, then DSQ_B is derived from DSQ_A after interchanging oriented numbers 1 and 3.

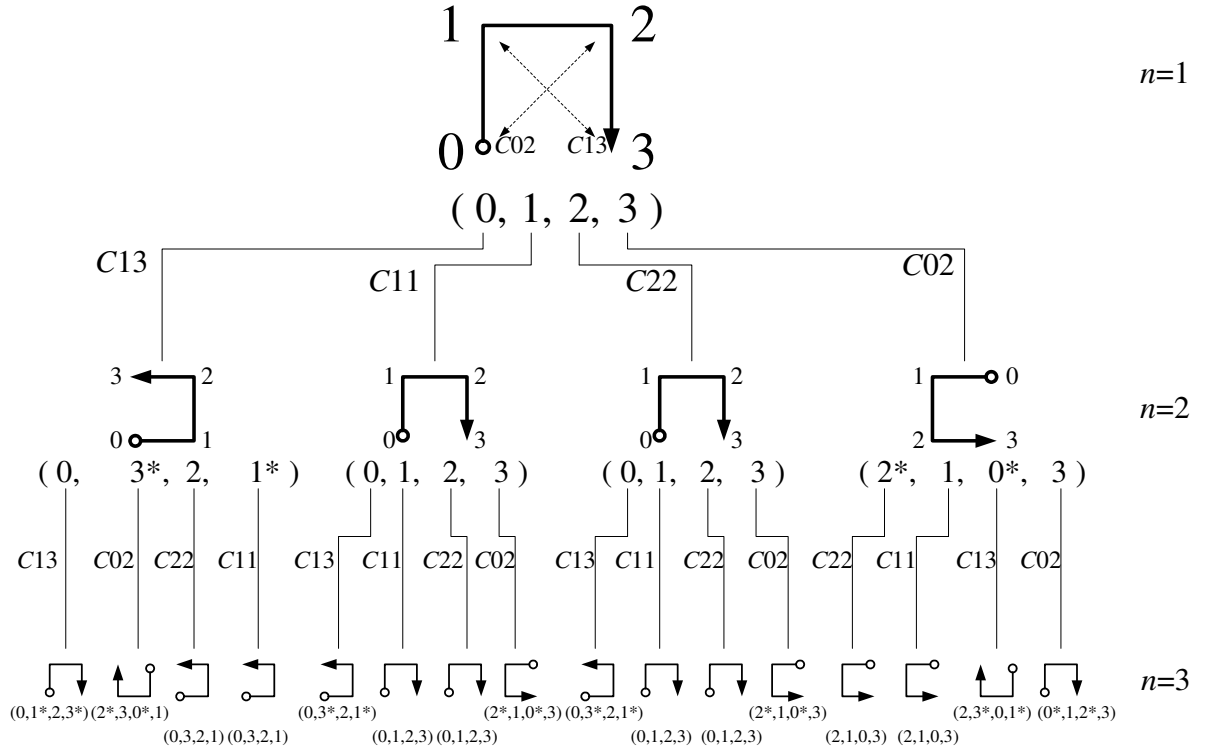
Case C02. If the a' th oriented number in block number $(A)_4$ is 3, then DSQ_B is derived from DSQ_A after interchanging oriented numbers 0 and 2.

Case C11/C22. If the a' th oriented number in block number $(A)_4$ is $1/2$, then DSQ_B is the same as DSQ_A .

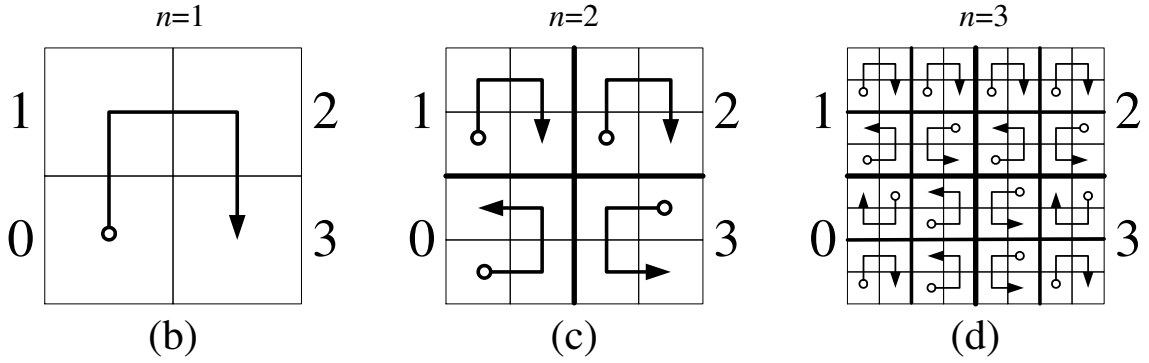
Take Figure 6-(d) as an example. Since direction sequences of four blocks in the Hilbert curve of order 1 are $(0, 1, 2, 3)$, direction sequence DSQ_3 of block $(3)_4$ is $(0, 1, 2, 3)$. In the Hilbert curve of order 2, the direction sequences for four sub-blocks in block $(\underline{3})_4$ are $(2^*, 1, 0^*, 3)$ derived from direction sequence $DSQ_3 = (0, 1, 2, 3)$ by Case C02. In the Hilbert curve of order 3, the direction sequences for four sub-blocks in block $(30)_4$ are $(2, 3^*, 0, 1^*)$ derived from direction sequence $DSQ_{30} = (2, 1, 0, 3)$ by Case C13. Figure 7-(a) shows all orientations of the Hilbert curve of order (n) 1, 2, and 3 which linearly order those equal-sized blocks in Figures 7-(b), (c), and (d), respectively. The corresponding direction sequence of each equal-sized block is shown below the orientation in Figure 7-(a).

3.2 The ONHC Strategy

In the ONHC strategy, the query point is distributed in the query block which is linearly ordered by the Hilbert curve of order n . There are two kinds of neighbors, inner neighbors and outer neighbors, which will be found by the ONHC strategy. Assume that the query block q in the curve of order n locates in block Q , block Q and block P are neighbors in the curve of order $n - 1$. Inner neighbors are equal-sized neighbors in the same block Q with the query block q . Outer neighbors are not in block Q . They are equal-sized neighbors in block P , or they are larger-sized than the query block q . Take block 50 in Figure 8 as an



(a)



C13: interchanging oriented numbers 1 and 3.


C02: interchanging oriented numbers 0 and 2.

C11/C22: no interchanging.


*: the changed oriented number after interchanging.

Figure 7: Orientations, direction sequences, and orders in the Hilbert curve: (a) all orientations and direction sequences in the curve of order (n) 1, 2, and 3; (b) the orientation of 4 equal-sized blocks in the curve of order 1; (c) the orientation of 16 equal-sized blocks in the curve of order 2; (d) the orientation of 64 equal-sized blocks in the curve of order 3.


21	22	25	26	37	38	41	42
20	23	24	27	36	39	40	43
19	18	29	28	35	34	45	44
16	17	30	31	32	33	46	47
15	12	11	10	53	52	51	48
14	13	8	9	54	55	50	49
1	2	7	6	57	56	61	62
0	3	4	5	58	59	60	63



query block



inner neighbor



outer neighbor

Figure 8: Example based on the Hilbert curve of order 3

example. Blocks 48, 49, and 51 are equal-sized inner neighbors, whereas blocks 52, 55, 56, 61 and 62 are equal-sized outer neighbors. Procedure *One_Neighbor_Finding* in Figure 9 shows the process of the ONHC strategy. In the initial state, we transform the sequence number of the query block into the block number in base 4. That is the formation of oriented numbers like $(d_1d_2 \dots d_i \dots d_n)_4$. Each d_i represents the oriented number in the Hilbert curve of order i . By these oriented numbers, we can generate direction sequences and obtain the locations in the Hilbert curve of different orders for the query block.

Take query block 50 in Figure 8 as an example. The process of the initial state is shown in Figure 10-(a). First, we transform sequence number $(50)_{10}$ into block number $(302)_4$ by Function *TransBase10to4*. The oriented numbers d_1 , d_2 , and d_3 are 3, 0, and 2, respectively. Then, we generate the direction sequences in the curve of different orders by Function *DSQ_Generation*. In the Hilbert curve of order 1, by oriented number $d_1 = 3$ in direction sequence $DSQ_3 = (0, 1, 2, \underline{3})$, block $(3)_4$ is the SE block of the whole space, as shown in Figure 10-(b). We store the location of the query block in the curve of order i in array $LQB[i]$. That is, $LQB[1] = 'SE'$. In the Hilbert curve of order 2, by oriented number $d_1 = 0$ and Case C02, direction sequence $DSQ_{30} = (2, 1, 0, 3)$ is derived from direction

```

Procedure One_Neighbor_Finding (block number  $h_{10}$ , order  $n$ , direction  $DirN$ )
01:   $h_4 := TransBase10to4(h_{10})$ ;
02:  /* Initial State:  $h_4 := (d_1 d_2 \dots d_i \dots d_n)_4$ ,  $d_i \in \{0, 1, 2, 3\}$ ,  $1 \leq i \leq n$ ; */
03:   $DSQ_{d_1} = (0, 1, 2, 3)$ ;   $LQB[1] = Block\_Location(d_1, DSQ_{d_1})$ ;
04:  For  $i = 2$  to  $n$  do    begin
05:     $X := (d_1 d_2 \dots d_i)_4$ ;   $Y := (d_1 d_2 \dots d_{i-1})_4$ ;
06:     $DSQ_X = DSQ\_Generation(d_{i-1}, d_i, DSQ_Y)$ ;
07:     $LQB[i] = Block\_Location(d_i, DSQ_X)$ ;  end;
08:  /* Local Finding */
09:  /* Array  $LQB$  stores the location of the query block in the curve of each order. */
10:  /* Array  $LNB$  stores the location of the neighboring block. */
11:  /* Variable  $findN$  stores the latest checked order in the local finding. */
12:   $findN = n$ ;   $flag = \text{'False'}$ ;   $flag = Local\_Finding(DirN, d_n, DSQ_h, n, LQB, LNB)$ ;
13:  /* Upward Finding */   $i = findN$ 
14:  While ( $flag = \text{'False'}$ ) and ( $i > 1$ )    begin
15:     $i = i - 1$ ;   $X := (d_1 d_2 \dots d_i \dots d_i)_4$ ;
16:     $flag = Local\_Finding(DirN, d_i, DSQ_X, i, LQB, LNB)$ ;   $findN = i$ ;  end;
17:  /* Downward Finding */   $i = findN$ ;
18:  While ( $flag = \text{'True'}$ ) and ( $i < n$ )    begin
19:     $i = i + 1$ ;   $LNB[i] = Neighbor\_Location(LQB[i], Reverse(DirN))$ ;  end;
20:  /* Block Number Generation */
21:  If ( $flag = \text{'True'}$ )    begin
22:     $X := (d_1 d_2 \dots d_i)_4$ ;   $Neighbor\_Generation(DirN, X, DSQ_X, findN, n, LNB)$ ;  end;
23:  else   $print(\text{'No neighbor exist.'})$ ;
end Procedure

```

Figure 9: Procedure *One_Neighbor_Finding*

sequence DSQ_3 . By oriented number $d_2 = 0$ in direction sequence $DSQ_{30} = (2, 1, \underline{0}, 3)$, block $(30)_4$ is the NE sub-block of block $(3)_4$, *i.e.*, $LQB[2] = 'NE'$, as shown in Figure 10-(c). In the Hilbert curve of order 3, by oriented number $d_2 = 0$ and Case C13, direction sequence $DSQ_{302} = (2, 3, 0, 1)$ is derived from direction sequence DSQ_{30} . By oriented number $d_3 = 2$ in direction sequence $DSQ_{302} = (\underline{2}, 3, 0, 1)$, block $(302)_4$ is the SW sub-block of block $(30)_4$, *i.e.*, $LQB[3] = 'SW'$, as shown in Figure 10-(d).

After the initial state, we start to do neighbor finding in direction $DirN$. Let's take the north ($DirN = 'N'$) neighbor finding of query block $(302)_4$ as an example. The process of the north neighbor finding is shown in Figure 10-(e). First, we find the north neighbor of query block $(302)_4$ in the Hilbert curve of order 3 by Function *Local_Finding*. By array $LQB[3]='SW'$, we find the north neighbor which is the NW sub-block of block $(30)_4$, as shown in Figure 10-(f). We also find another NE block which also in the north of block $(30)_4$. The NE block is the north west neighbor of the query block. We store the location of these two neighbors in array LNB , *i.e.*, $LNB[3][1]='NW'$ and $LNB[3][2]='NE'$. At this point, Function *Local_Finding* returns 'True'. It means that we find two equal-sized inner neighbors in the curve of order 3. Finally, we can generate their block numbers by Procedure *Neighbor_Generation*. By array $LNB[3][1]='NW'$ and $LNB[3][2]='NE'$, and direction sequence $DSQ_{30Z} = (2, \underline{3}, \underline{0}, 1)$, two oriented numbers which represent the NW and NE sub-blocks of block $(30)_4$ are 3 and 0, respectively. Therefore, two inner neighbors are blocks $(303)_4$ and $(300)_4$ which can be transformed into sequence numbers 51 and 48, respectively, as shown in Figure 10-(g).

When we cannot find the inner neighbor in direction $DirN$ in the curve of order n , we continue to find the outer neighbor in the block which locates in the curve of order $n - 1$ by checking the oriented number d_{n-1} . If the outer neighbor still can not be found by the oriented number d_{n-1} , we continue to find it in the block which locates in the curve of order $n - i$ with increasing the value of variable i ($i < n$) by one. If we still cannot find the outer neighbor in the block of the curve of order 1, it means that no neighbor exists in direction $DirN$.

Take the south neighbor finding of query block $(302)_4$ in Figure 10-(d) as an example. By array $LQB[3]='SW'$ of query block $(302)_4$, the south neighbor can not be found in

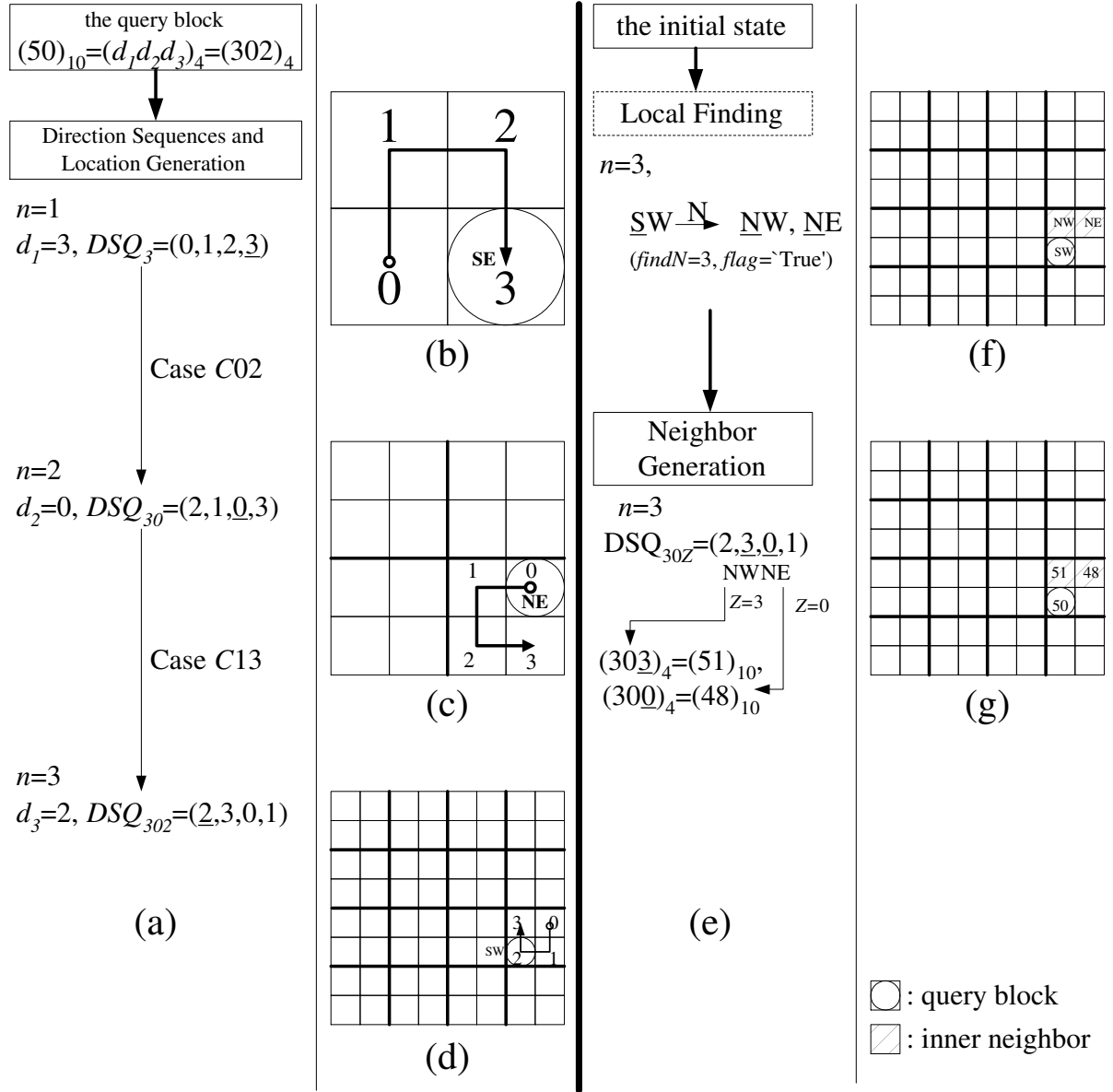


Figure 10: The initial state and north neighbor finding for block $(50)_{10} = (302)_4$: (a) the process of the initial state; (b) block $(3)_4$ in the Hilbert curve of order 1; (c) block $(30)_4$ in the Hilbert curve of order 2; (d) block $(302)_4$ in the Hilbert curve of order 3; (e) the process of north neighbor finding; (f) locations in the curve of order 3; (g) sequence numbers in the curve of order 3.

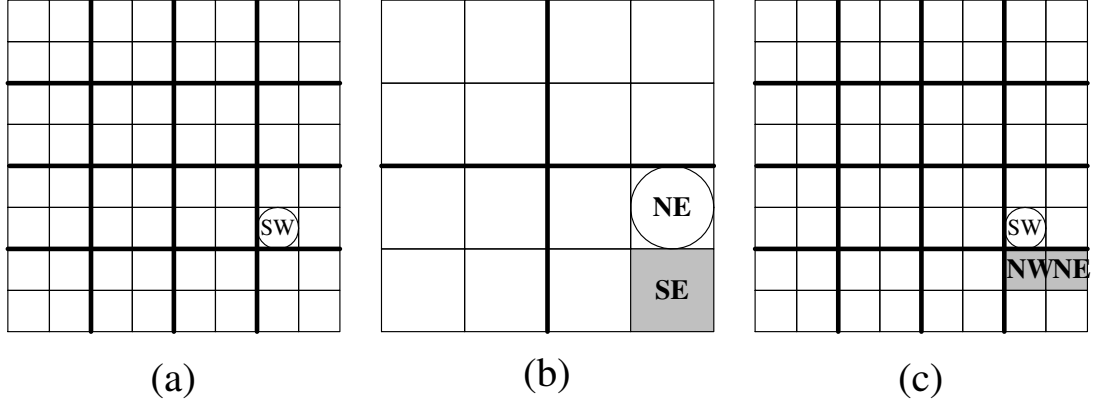


Figure 11: The south neighbor finding for block $50_{10} = (302)_4$: (a) the local finding in the curve of order 3; (b) the upward finding in the curve of order 2; (c) the downward finding in the curve of order 3.

the curve of order 3 and Function *Local_finding* returns 'False', as shown in Figure 11-(a). Therefore, we do the upward finding by array $LQB[2]=\underline{\text{NE}}$, as shown in Procedure *One_Neighbor_Finding* in Figure 9. That is, we perform Function *Local_Finding* in the curve of order 2 to find the south neighbor of block $(30)_4$, as shown in Figure 11-(b). By $LQB[2]=\underline{\text{NE}}$, the south neighbor of block $(30)_4$ can be found as the $\underline{\text{SE}}$ block, *i.e.*, $LN B[2]=\text{'SE'}$. At this point, Function *Local_Finding* returns 'True' which means the south outer neighbor is found in the curve of order 2. Array $LN B[1] = LQB[1] = \text{'SE'}$. The value of variable *findN* in Procedure *One_Neighbor_Finding* is equal to 1.

Next, we do the downward finding to find the south outer neighbor in the curve of the same order with the query block, as shown in Procedure *One_Neighbor_Finding* in Figure 9. We start the downward finding in the curve of order equal to *findN*. Since the south outer neighbor is found in the $\underline{\text{SE}}$ block in the curve of order 2, the south outer neighbor would be in the north of SE block, as shown in Figure 11-(c). Therefore, we find the location of the north outer neighbor in curve of order i ($i > \textit{findN}$ and $i < n$) by the location of the query block in the curve of order i ($LQB[i]$) and the reverse of *DirN* (*i.e.*, $\textit{Reverse}(\textit{DirN})$). We can obtain the location of the south outer neighbor in the curve of order 3 by $LQB[3] = \underline{\text{'SW'}}$ and $\textit{Reverse}(\textit{DirN})=\text{'N'}$, *i.e.*, $LN B[3] = \underline{\text{'NW'}}$. We also find another north west outer neighbor: $\underline{\text{NE}}$ sub-block in the curve of order 3 of the SE block in the curve of order 2, *i.e.*, $LN B[3][2] = \text{'NE'}$.

Finally, we can generate block numbers of these two north outer neighbors by Procedure *Neighbor_Generation*. By array $LNB[1] = 'SE'$ and direction sequence $DSQ_X = (0, \underline{1}, 2, 3)$, we know that oriented number 3 ($= X$) represents the SE block in the curve of order 1. Direction sequence $DSQ_{3Y} = (2, 1, 0, 3)$ is derived from DSQ_3 by oriented number 3 and Case *C02*. By array $LNB[2] = 'SE'$ and direction sequence $DSQ_{3Y} = (2, 1, 0, \underline{3})$, we know that oriented number 3 ($= Y$) represents the SE sub-block of block $(3)_4$ in the curve of order 2. Direction sequence $DSQ_{33Z} = (0, 1, 2, 3)$ is derived from DSQ_{33} by oriented number 3 and Case *C02*. By array $LNB[3][1] = 'NW'$, array $LNB[3][2] = 'NE'$, and direction sequence $DSQ_{33Z} = (0, \underline{1}, \underline{2}, 3)$, we know that oriented numbers 1 and 2 represent the NW and NE sub-blocks of block $(33)_4$, respectively, in the curve of order 3. Two north outer neighbors are blocks $(331)_4$ and $(332)_4$ and are transformed into sequence numbers 61 and 62, respectively, as shown in Figure 8.

Therefore, we can use the quaternary number of the query block and the direction sequence to obtain the relative location of the neighbor in the curve of each order. There are eight neighbors in eight directions to query block 50 in Figure 8. For the other query blocks in Figure 8, the number of the neighbors is less than eight since they locate in the side or corner of the space. For example, the number of neighbors of query block 42 is 3.

3.3 The ANHC Strategy

Let A and B be two spatial datasets and $dist(a, b)$ be a distance metric. Then, the *all-nearest-neighbors* query is defined as: $ANN(A, B) = \{ \langle a_i, b_j \rangle : \forall a_i \in A, \exists b_j \in B, \neg \exists b_k \in B \{ dist(a_i, b_k) < dist(a_i, b_j) \} \}$ [31]. In other words, the query finds nearest neighbor(s) in B for each object in A . We assume that two datasets A and B are in the data space of the same range of the coordinate system. Because of different sizes of datasets A and B , datasets A and B are organized by the Hilbert curve of orders n_A and n_B , respectively. That is, for each data a_i in dataset A (or b_i in dataset B) has its h_A (or h_B) value in base 10 which is the sequence number in the Hilbert curve of order n_A (or n_B).

Figure 12 shows our *ANHC* strategy. In the all-nearest-neighbors finding $ANN(A, B)$, we sort sequence numbers (h_A values) in dataset A in Step 1 (Line 02 of Figure 12). In Step 2, for each query block with sequence number h_A , we discuss three conditions of $ANN(A, B)$.

B) depending on two orders n_A and n_B as follows.

Condition 1. When order n_A is equal to order n_B , it means that the size of the block with sequence number h_A in dataset A is equal to that with sequence number h_B in dataset B . We denote sequence number h_C as the candidate value in the curve of order n_B . It is equal to sequence number h_A . We use our *ONHC* strategy (*i.e.*, Procedure *One_Neighbor_Finding*) to find eight sequence numbers by sequence number h_C , order n_B , and direction $DirN$ in dataset B . These eight sequence numbers and sequence number h_C represents the candidate neighboring blocks in dataset B . We store these candidate sequence number in dataset $CNNSet$. The corresponding process is shown from Lines 09 to 13 of Figure 12.

Condition 2. When order n_A is larger than order n_B , it means that the size of the block with sequence number h_A in dataset A is smaller than that with sequence number h_B in dataset B . We first transform sequence number h_A value into sequence number h_C by the equation $h_C = h_A * 4^{n_B - n_A}$. Then, we use our *ONHC* strategy (*i.e.*, Procedure *One_Neighbor_Finding*) to find eight sequence numbers by sequence number h_C , order n_B , and direction $DirN$ in dataset B . These eight sequence numbers and sequence number h_C represents the candidate neighboring blocks in dataset B . Finally, we store these candidate sequence numbers in dataset $CNNSet$. The corresponding process is shown from Lines 14 to 19 of Figure 12.

Condition 3. When order n_A is smaller than order n_B , it means that the size of the block with sequence number h_A in dataset A is larger than that with sequence number h_B in dataset B . We first use our *ONHC* strategy (*One_Neighbor_Finding*) to obtain eight sequence numbers in dataset A and store them in dataset $TempSet$, including sequence number h_A . Then, we transform each sequence number h_t in dataset $TempSet$ into sequence number h_C by equation $h_C = h_t \times 4^{n_B - n_A}$. The number of sequence number h_C for each sequence number h_t is $4^{n_B - n_A}$. Finally, we store these candidate sequence numbers in dataset $CNNSet$. The corresponding process is shown from Lines 20 to 29 of Figure 12.

In Step 3 (Line 30 of Figure 12), we sort sequence numbers in dataset $CNNSet$ and

Procedure *All_Nearest_Neighbors_Finding* (DataSet A , DataSet B , Order n_A , Order n_B)

```

01:  begin
02:     $ANNSet := \phi$ ; /*  $ANNSet$  is the result of  $ANN(A, B)$ . */
03:    Sorting sequence numbers  $h_A$  in dataset  $A$ ; /* Step 1 */
04:    /* Data  $a_i$  locates at the block with sequence number  $h_A$  in the curve of order  $n_A$ . */
05:    For each data  $a_i$  with sequence number  $h_A$  in dataset  $A$  do /* Step 2 */
06:      begin
07:         $CNNSet := \phi$ ;
08:        /*  $CNNSet$  is the candidate set to store candidate neighbors in dataset  $B$ . */
09:        if  $(n_A - n_B) = 0$  then /* Condition 1 */
10:          begin  $h_C = h_A$ ;  $CNNSet = CNNSet \cup \{h_C\}$ ;
11:            For each  $DirN \in \{'S', 'N', 'E', 'W'\}$  do
12:               $CNNSet = CNNSet \cup One\_Neighbor\_Finding(h_C, n_B, DirN)$ ;
13:            end;
14:          if  $(n_A - n_B) > 0$  then /* Condition 2 */
15:            begin  $h_C = h_A \times 4^{n_B - n_A}$ ;  $CNNSet = CNNSet \cup \{h_C\}$ ;
16:              For each  $DirN \in \{'S', 'N', 'E', 'W'\}$  do
17:                 $CNNSet = CNNSet \cup One\_Neighbor\_Finding(h_C, n_B, DirN)$ ;
18:              end;
19:            if  $(n_A - n_B) < 0$  then /* Condition 3 */
20:              begin
21:                 $TempSet := \{h_A\}$ ;
22:                /*  $TempSet$  is the temporary set to store candidate neighbors in dataset  $A$ . */
23:                For each  $DirN \in \{'S', 'N', 'E', 'W'\}$  do
24:                   $TempSet = TempSet \cup One\_Neighbor\_Finding(h_A, n_A, DirN)$ ;
25:                For each  $h_t \in TempSet$  do
26:                   $CNNSet = CNNSet \cup$ 
27:                     $\{h_C \mid h_C \text{ ranges from } h_C = h_t \times 4^{n_B - n_A} \text{ to } h_C = h_t \times 4^{n_B - n_A} + 4^{n_B - n_A} - 1\}$ ;
28:                end;
29:              Sorting and Filtering distinct sequence numbers  $h_c$  in  $CNNSet$ ; /* Step 3 */
30:              For each  $h_C$  in  $CNNSet$  do
31:                begin
32:                  if the block with sequence number  $h_c$  has no data ;
33:                  else For each data  $b_j$  in the block with sequence number  $h_B$  in data set  $B$  do
34:                     $NN := \{b_j : \exists b_j \in B, \neg \exists b_k \in B \{dist(q, b_k) < dist(q, b_j)\}\}$ 
35:                  end;
36:                 $ANNSet := ANNSet \cup \{(a_i, NN)\}$ ;
37:              end;
38:            end procedure

```

Figure 12: Procedure *All_Nearest_Neighbors_Finding*

remove the duplicate sequence numbers. After sorting, we can retrieve the block from the disk once by the sequence order. Then, from Lines 31 to 36 of Figure 12, we check the block with each sequence number h_C in dataset B . We access those blocks which has data in dataset B and retrieve the data from them. Next, we make the comparison between data a_i in block h_A in dataset A and data b_k in block h_B in dataset B . Finally, we can obtain the nearest neighbor b_j in data set B for each a_i in dataset A . The final result of all-nearest-neighbors query $\text{ANN}(A, B)$ is ANNSet which includes pairs of (a_i, b_j) (Line 37 of Figure 12).

Take Figure 13 as an example. Figure 13-(a) shows that two datasets A and B are ordered by the Hilbert curve of order 2 (n_A) and 3 (n_B), respectively. The gray-colored blocks shows that there exists the data points. In Step 1, we sort h_A values in dataset A . In step 2, since order n_A is smaller than order n_B , it is Condition 3 in our ANHC strategy. Take query block 7 in Figure 13-(b) as an example. We use our ONHC strategy to obtain sequence numbers of eight neighboring blocks, *e.g.*, sequence number 2 of the south neighbor. All sequence numbers of neighbors are stored as candidate neighbors (h_t values) in dataset TempSet . The number of neighbors for the different query block is different depending on the location of query block. For example, query block 0 in Figure 13-(a) only has three neighbors, because it locates in the corner. Then, we transform each sequence number in dataset TempSet into four candidate neighbors (h_C values) in dataset CNNSet . For example, we transform sequence number 2 into four sequence numbers 8, 9, 10, and 11, respectively, by four equations $2 * 4^{(3-2)}$, $2 * 4^{(3-2)} + 1$, $2 * 4^{(3-2)} + 2$, and $2 * 4^{(3-2)} + 3$, respectively, as shown in 13-(b). After sorting and filtering in Step 3, we need the small number of disk access to retrieve the nearest neighbor from sequential ranges of candidate neighbors (h_C values) in dataset B . For example, the number of disk access for query block 7 is twice which is derived from two ranges of h_C values. One ranges from 8 to 39, and another ranges from 52 to 55 in dataset B , as shown in Figure 13-(a).

For the all-nearest-neighbors finding of $\text{ANN}(A, B)$ in Figure 14-(a), two datasets A and B are ordered by the Hilbert curve of order 3 (n_A) and 2 (n_B), respectively. In Step 1, we sort h_B values in dataset B . In Step 2, since order n_B is larger than order n_A , it is Condition 2 in our ANHC strategy. Take query block 28 in Figure 14-(a) as an example.

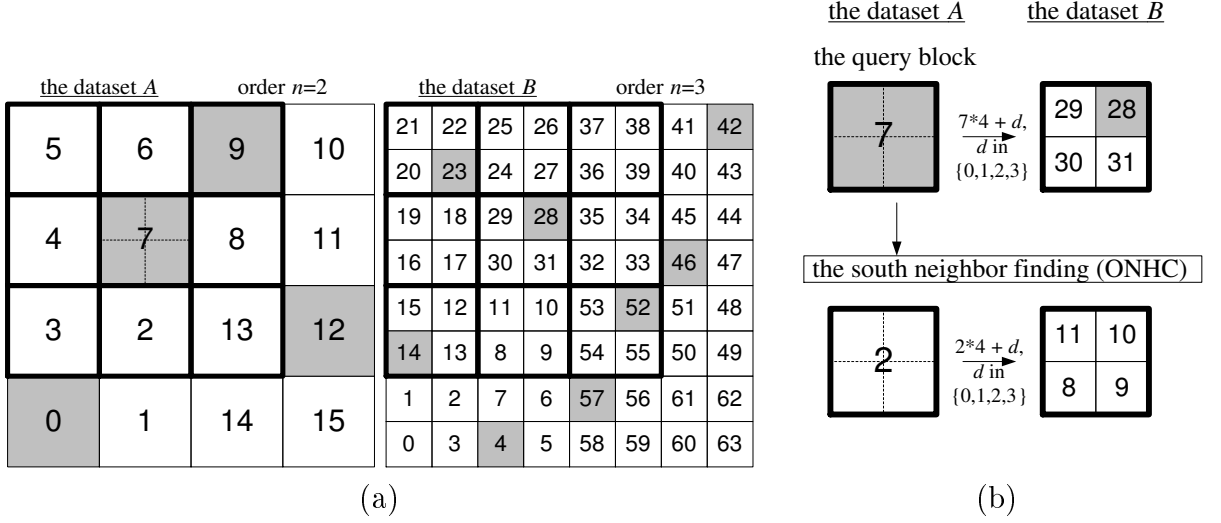


Figure 13: An example of the ANN query, $\text{ANN}(A, B)$, where $n_A < n_B$: (a) two datasets *A* and *B*; (b) the south neighbor finding by the ONHC strategy.

We transform sequence number 28 into sequence number 7 by equation $28 * 4^{(2-3)}$ and store sequence number 7 in the dataset *CNNSet*. Then, we use our ONHC strategy to obtain sequence numbers of eight neighboring blocks by sequence number 7. All sequence numbers of neighbors are stored as candidate neighbors (h_C values) in dataset *TempSet*. After sorting and filtering in Step 3, we only need the small number of disk access to retrieve the nearest neighbor from sequential ranges of candidate neighbors (h_C values) in dataset *A*. For example, the number of disk accesses for query block 28 is twice which is derived from two ranges of h_C values. One ranges from 2 to 9, and another is 13 in dataset *B*, as shown in Figure 14-(a).

4 The Performance Study

In this section, we make two comparisons. The first one is the comparison of our ONHC strategy and the CCSF strategy [5] on the one-nearest-neighbor query. The second one is the comparison of our ANHC strategy and the strategy based on R^* -tree (the variation of the R -tree) [3, 7, 31] on the all-nearest-neighbors query.

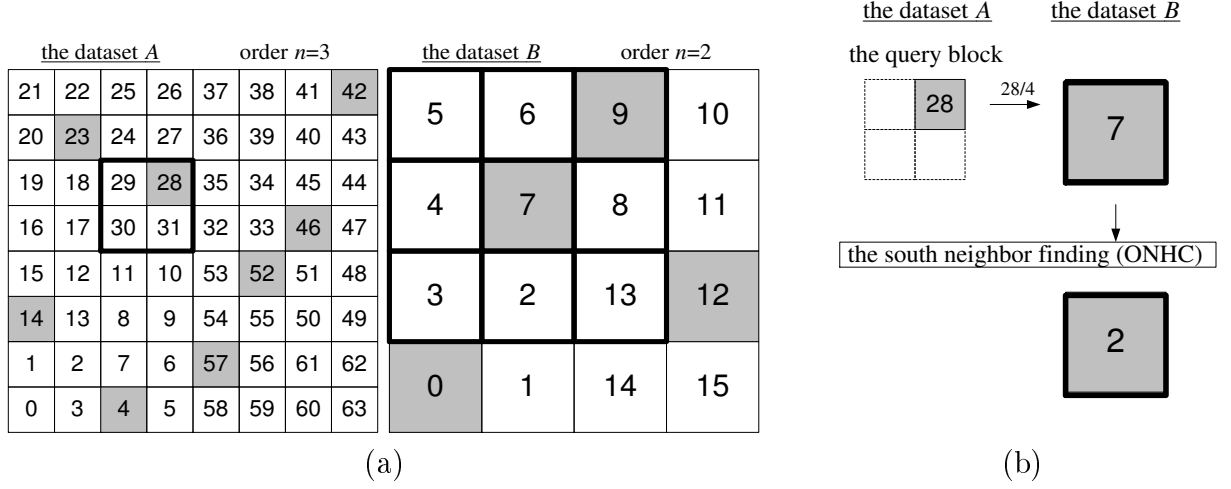


Figure 14: An example of the ANN query, $\text{ANN}(A, B)$, where $n_A > n_B$: (a) two datasets *A* and *B*; (b) the south neighbor finding by the ONHC strategy.

4.1 The Performance Model

Our simulation is implemented in Java and the experiments are executed using a PC with a Pentium M 735 processor, 256 MB DDR, 7200 rpm IBM Hard Disk, and running Windows XP. We consider the CPU time and the I/O time as our performance measures. The CPU time means the time to compute the location of the nearest neighbor. The I/O time, which means the time to access a block, consists of two parts: *positioning time* (PT) and *transfer time* (TT) [26]. The positioning time is the time to move to the right position in the disk, which includes the seek time and rotational delays. The transfer time is simply the time to transfer the requested block from the disk into main memory.

We focus on the point data and the experimental analysis is carried out on six distinct spatial datasets. Figure 15 shows these six cases of data distribution which are described as follows [25].

- (a) **Uniform Distribution:** The data objects are uniformly distributed in the overall data space.
- (b) **Centralized Distribution:** Most of the data objects are centralized in a small region.

- (c) **Diagonal Distribution:** The data objects follow a uniform distribution along the main diagonal.
- (d) **X-parallel Distribution:** Most of the data objects are located on a line which is parallel to the X-axis.
- (e) **Sine Distribution:** The data objects follow a sine curve.
- (f) **Real Dataset:** The data objects of 30674 streets and 17790 utility network elements are download from R-tree Portal (<http://www.rtreeportal.org>).

Each dataset contains 10000 data points in the $2D$ -space which is 10000×10000 . For each dataset in Figures 15-(a) to (e), 100 files of 10000 query points are randomly generated to make the nearest neighbor queries. In the following comparisons, for the strategy based on the Hilbert curve, data points are assigned to the corresponding disk block by their h -values. The points in the same disk block are linked. The capacity of each disk block, denoted as *block_capacity*, is assigned to be 10 points. Because of the limitation of *block_capacity*, a Hilbert curve of different order p ($p > 1$) is needed for the different datasets, as shown in the second row of Table 16. For example, for the uniform data distribution in Figure 15-(a), the Hilbert curve of order 5 is needed to order 10000 data points. The reason is that we use 1024 ($= 2^5 \times 2^5$) disk blocks to store these points such that each disk block contains 10 points on the average. For the X-parallel, diagonal, centralization, and sine data distributions in Figures 15-(b), (c), (d), and (e), respectively, most of their data points are distributed in the certain region or near the certain line. Because of the limitation of *block_capacity*, the Hilbert curve of order (5 or 6 or 7) is needed for each of these datasets. For two real datasets like Figure 15-(f), the Hilbert curve of order 6 is used for them.

For the strategy based on the R^* -tree, the capacity of the node is assigned to be 10 points. Each leaf node entry contains the coordinates of a point. Since the sequence of inserting data points affects the structure of the R^* -tree, we sort and group the data points by using the Hilbert curve on building and processing an R^* -tree for datasets and use the pruning metrics in [7, 31]. We take the average of the CPU-time and the I/O time of the ANN query for each dataset as shown in Figure 15.

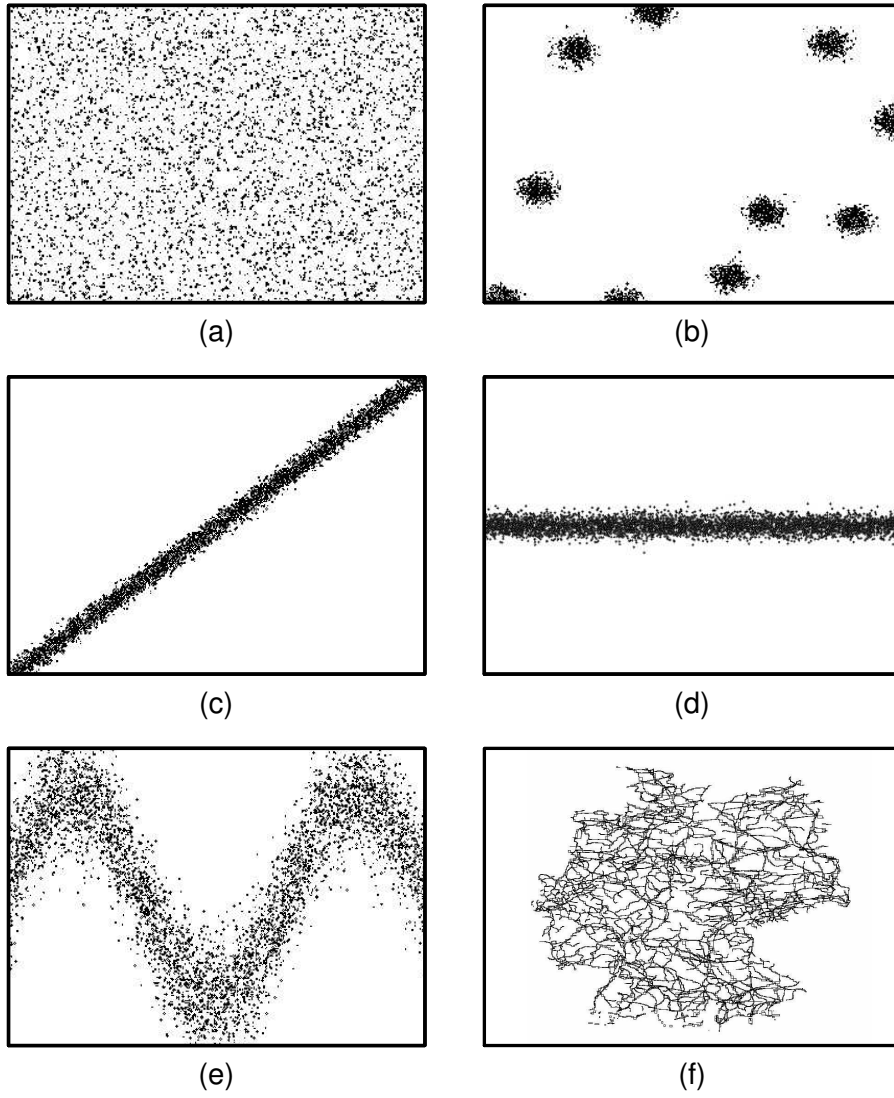


Figure 15: Five cases of data distribution: (a) uniform; (b) centralized; (c) diagonal; (d) X-parallel; (e) sine; (f) real dataset.

Measure	Data Distribution	Uniform	Centralized	Diagonal	X-Parallel	Sine
CPU time	CCSF	1034	1291	1805	1032	1792
	ONHC	175	214	291	179	273

Figure 16: A comparison of our ONHC strategy and the CCSF strategy (milliseconds)

4.2 Simulation Results

In the first comparison of our ONHC strategy and the CCSF strategy [5], the result is shown in Table 16. Our strategy needs less CPU time than the CCSF strategy for all different datasets. The reason is that our strategy computes the sequence numbers of the eight neighboring blocks directly based on Hilbert curve. It is different from the CCSF strategy which takes most of the CPU-time on the complicated transformation rules between the Peano curve and the Hilbert curve to obtain the eight neighboring blocks.

After obtaining the sequence numbers of eight neighboring blocks, our strategy can directly access these blocks. The I/O time can be reduced without accessing the unnecessary blocks. Since it is more efficient to fetch a set of consecutive disk blocks than a randomly scattered set, the additional positioning time [5, 14, 16, 21, 20] can be reduced based on the good clustering property of the Hilbert curve. Since our ONHC strategy and CCSF strategy are both based on the Hilbert curve, both I/O time of our ONHC strategy and CCSF strategy for each one of five data distributions are always equal to $(4*PT+8*TT)$.

In the second comparison, we consider two variables: the data count and the data distribution, which affect the performance of the ANN query. Since the ANN query of dataset A is performed on dataset B , we evaluate the performance of the ANN query in four cases, as shown in Figure 17. For the parameter of the data count, the symbol ‘*1’ denotes the variable data count and the blank denotes the static data count. The data count ranges from 10^3 to 10^4 data points. For the parameter of the data distribution, the symbol ‘*2’ denotes the variable data distribution and the blank denotes the uniform data distribution. Take Case C1 as an example. The data count of dataset A is variable, whereas the data count of dataset B is static. Datasets A and B are both created in the uniform distribution.

Figure 18 shows the result of Case C1. Our ANHC strategy needs less I/O time and CPU time than the strategy based on the R^* -tree as the data count of dataset A increases,

Parameter Case	Data Count		Data Distribution	
	A	B	A	B
C1	*1			
C2		*1		
C3			*2	
C4				*2
C5	*3	*4		
C6	*4	*3		

*1: the variable data count.

*2: the variable data distribution.

*3: the real dataset of 30674 objects.

*4: the real dataset of 17790 objects.

Figure 17: Five cases ($C1$, $C2$, $C3$, $C4$, $C5$) of the comparison on the ANN query for datasets A and B

as shown in Figures 18-(a) and (b). The reason is that large number of nodes in the R^* have to be compared and accessed to obtain the actual nearest neighbor of one point. On the other hand, in our ANHC strategy, the 1000 data points, 2000 to 9000 data points, and 10000 data points are distributed in 4^4 , 4^5 , and 4^6 sequence numbers, respectively, because of limitation of *block_capacity*. These sequence numbers are linearly ordered by the Hilbert curve of order 4, 5, and 6, respectively. Our ANHC strategy directly computes eight sequence numbers of neighboring blocks next to one query block Q . Therefore, the I/O time and CPU time of our ANHC strategy are related with the number of sequence numbers, instead of the data count. Because the order N_A of dataset A is smaller than or equal to the one N_B of dataset B , $8 * 4^{(N_B - N_A)}$ neighboring blocks in dataset B are needed to be found for the query block in dataset A . Based on the good clustering property of the Hilbert curve, these $4^{(N_B - N_A)}$ neighboring blocks in one of eight direction are linearly ordered in the disk. Therefore, the I/O time can be reduced by accessing these blocks once. Moreover, our strategy directly accesses these eight neighboring blocks next to the query block Q to compare the distance between data points. Once the nearest neighbor of the query block Q is found in these eight neighboring blocks, our ANHC strategy stops finding the other neighboring blocks which are larger than the size of these eight neighboring blocks.

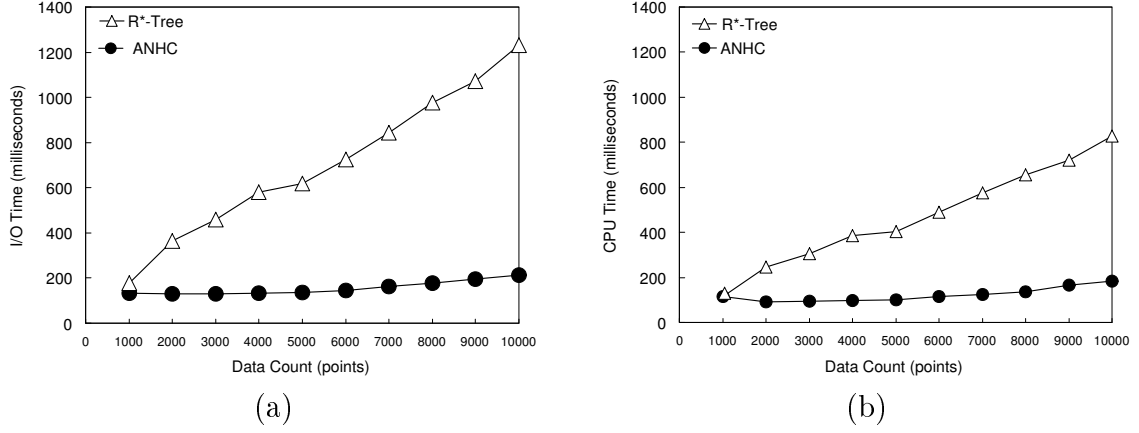


Figure 18: A comparison of Case C1: (a) I/O time; (b) CPU Time.

Figure 19 shows the result of Case C2. Our ANHC strategy needs less I/O time and CPU time than the strategy based on the R^* -tree as the data count of dataset B increases, as shown in Figures 19-(a) and (b). The reason is that the R^* -trees with different height are built for the increasing data count of dataset B . The tree heights to build R^* -trees for 1000 points, 2000 to 5000 points, 6000 to 10000 points are 4, 5, and 6, respectively. The performance of the same query count of dataset A is affected by the tree height of R^* -tree. It means that the increasing number of nodes in the R^* -tree have to be compared and accessed to obtain the actual nearest neighbor of one query point as the height increases. Then, the I/O time and CPU time increase. On the other hand, the I/O time and CPU time of our ANHC strategy are related with the number of the sequence numbers for the neighboring blocks, instead of the data count. Although the order N_B of dataset B is smaller than or equal to the one N_A of dataset A , the number of neighboring blocks in dataset B for the *query* block in the dataset A remains eight. Therefore, the I/O time and CPU time of our ANHC strategy increases a little, because the number of data points in one block increases as the data count of dataset B increases.

Figure 20 shows the result of Case C3. Our ANHC strategy needs less I/O time and CPU time than the one based on the R^* -tree for different kinds of data distribution. The reason is that the skew data distribution of dataset A , as shown in Figures 15-(b) to (e), affects the height and the number of nodes in the R^* -tree for dataset A . Although dataset

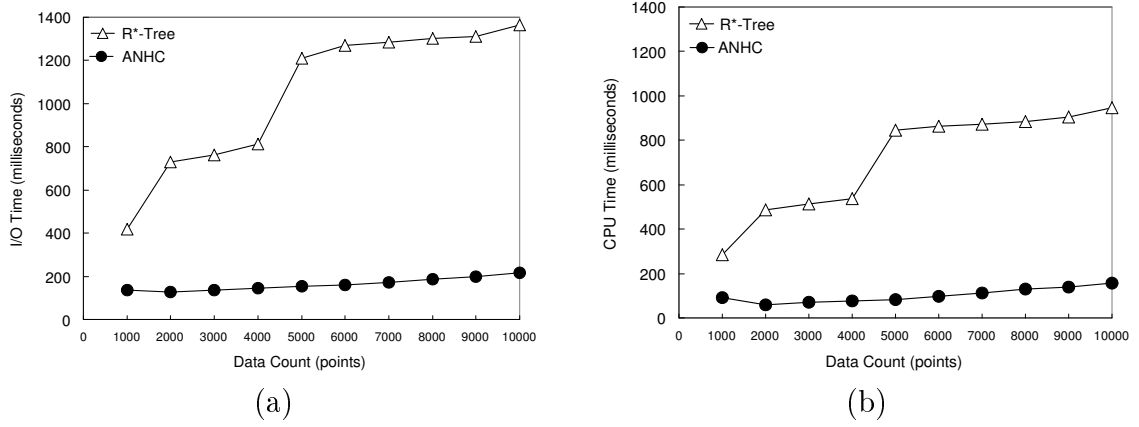


Figure 19: A comparison of Case C2: (a) I/O time; (b) CPU Time.

Measure \ Data Distribution		Uniform	Centralized	Diagonal	X-Parallel	Sine
I/O time	R-tree	8732	9375	10795	8877	8240
	ANHC	235	64	110	45	73
CPU time	R-tree	2939	3216	3872	3222	2752
	ANHC	64	48	48	44	46

Figure 20: A comparison of Case C3 (milliseconds)

B is uniformly distributed, as shown in Figure 15-(a), large number of nodes are needed to be compared and accessed to obtain the actual nearest neighbor of one query point. On the other hand, in our ANHC strategy, datasets A and B which have the same data count are both distributed in blocks of 4^6 sequence numbers. These sequence numbers are linearly ordered by the Hilbert curve of order 6. Our ANHC strategy can directly compute and access eight neighboring blocks in dataset B next to the query block Q in dataset A . Figure 21 shows the result of Case C4. Our ANHC strategy needs less I/O time and CPU time than the strategy based on the R^* -tree for the different data distribution. The reason is the same as that for the result of case C3. In addition, our strategy directly accesses at least one neighboring blocks in dataset B next to the query block Q in dataset A . Figure 22 shows the result of Cases C5 and C6. Our ANHC strategy needs less I/O time and CPU time than the strategy based on the R^* -tree for two real datasets. The reason is the same as that for the result of cases C1 and C2.

Therefore, our ANHC strategy can directly compute and access eight neighboring blocks

Measure \ Data Distribution		Uniform	Centralized	Diagonal	X-Parallel	Sine
I/O time	R-tree	8732	1358	6115	3852	3688
	ANHC	235	668	422	707	579
CPU time	R-tree	2939	291	2108	1453	1154
	ANHC	64	291	82	118	168

Figure 21: A comparison of Case C4 (milliseconds)

Measure \ Case		C5	C6
I/O time	R-tree	15063	9890
	ANHC	4641	4547
CPU time	R-tree	10377	7064
	ANHC	3627	3668

Figure 22: A comparison of Cases C5 and C6 (milliseconds)

next to the query block, instead of unnecessary nodes accessing by the strategy based R^* -tree. No matter whether the data count or the data distribution is variable, our strategy based on the Hilbert curve needs less I/O and CPU time than the strategy based on the R^* -tree.

5 Conclusion

In this paper, we have presented the ONHC strategy to answer the ONN query. We first generated direction sequences to store the orientations of the query block in the Hilbert curve of different orders. By using quaternary numbers and direction sequences of the query block, we obtained the relative locations of the neighboring blocks and computed their quaternary numbers. Finally, we can directly access the neighboring blocks by their sequence numbers which is the transformation of the quaternary numbers from base four to ten. The nearest neighbor can be obtained by comparisons in these blocks. Then, we have presented the ANHC strategy to answer the all-nearest-neighbors query by using our ONHC strategy. Finally, we have compared our ONHC strategy and ANHC strategy with the CCSF strategy and the strategy based on R -trees, respectively. In the performance of the response time (CPU time and I/O time), our ONHC strategy needs less time than the CCSF strategy for the one-nearest neighbor query. Our ANHC strategy needs less time than the strategy based on R -trees for the all-nearest neighbors query.

References

- [1] J. M. Bahi and A. Mostefaoui, “Locational and Coverage for High Density Sensor Networks,” *Computer Communications*, Vol. 31, No. 4, pp. 770–781, March 2008.
- [2] J. J. Bartholdi and P. Goldsman, “Vertex-Labeling Algorithms for the Hilbert Space-filling Curve,” *Software-Practice and Experience*, Vol. 31, No. 5, pp. 395–408, April 2001.
- [3] N. Beckmann, H. P. Kriegel, R. Schneider and B. Seeger, “The R*-tree: An Efficient and Robust Access Method for Points and Rectangles,” *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pp. 322–331, 1990.
- [4] J. Castro, M. Georgiopoulos, R. Demara and A. Gonzalez, “Data-Partitioning Using the Hilbert Space Filling Curves: Effect on the Speed of Convergence of Fuzzy ARTMAP for Large Database Problems,” *Neural Networks*, Vol. 18, No. 7, pp. 967–984, Sept. 2005.
- [5] H. L. Chen and Y. I. Chang, “Neighbor Finding Based on Space Filling Curves,” *Information Systems*, Vol. 30, No. 3, pp. 205–226, May 2005.
- [6] N. Chen, N. Wang and B. Shi, “A New Algorithm for Encoding and Decoding the Hilbert Order,” *Software-Practice and Experience*, Vol. 37 No. 8, pp. 897–908, July 2007.
- [7] Y. Chen and J. M. Patel, “Efficient Evaluation of All-Nearest-Neighbor Queries,” *Proc. of the 23rd Int. Conf. on Data Eng.*, pp. 1056–1065, 2007.
- [8] K. L. Chung, Y. L. Huang and Y. W. Liu, “Efficient Algorithms for Coding Hilbert Curve of Arbitrary-sized Image and Application to Window Query,” *Information Sciences*, Vol. 177, No. 10, pp. 2130–2151, May 2007.
- [9] X. Deng, S. Reyner, X. Liu, Q. Zhang, Y. Yang and N. Li, “DHPC: A New Tool to Express Genome Structural Features,” *Genomics*, Vol. 91, No. 5, pp. 476–483, May 2008.

- [10] S. F. Frisken and R. N. Perry, "Simple and Efficient Traversal Methods for Quadrees and Octrees," *Journal of Graphics Tools*, Vol. 7, No. 3, pp. 1–11, 2002.
- [11] D. Guo and M. Gahegan, "Spatial Ordering and Encoding for Geographic Data Mining and Visualization," *Journal of Intelligent Information Systems*, Vol. 27, No. 3, pp. 243–266, Nov. 2006.
- [12] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pp. 47–57, 1984.
- [13] H. V. Jagadish, "Linear Clustering of Objects with Multiple Attributes," *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, Vol. 19, No. 2, pp. 332–342, 1990.
- [14] H. V. Jagadish, "Analysis of the Hilbert Curve for Representing Two-Dimensional Space," *Information Processing Letters*, Vol. 62, No. 1, pp. 17–22, April 1997.
- [15] N. Koudas, "Indexing Support for Spatial Joins," *Data and Knowledge Eng.*, Vol. 34, No. 1, pp. 99–124, 2000.
- [16] J. K. Lawder and P. J. H. King, "Querying Multi-dimensional Data Indexed Using the Hilbert Space-Filling Curve," *ACM SIGMOD Record*, Vol. 30, No. 1, pp. 19–24, 2001.
- [17] J. Y. Liang, C. S. Chen, C. H. Huang and L. Liu, "Lossless Compression of Medical Images Using Space-Filling Curves," *Computerized Medical Imaging and Graphics*, Vol. 32, No. 3, pp. 174–182, April 2008.
- [18] S. Liao, M. A. Lopez, and S. T. Leutenegger, "High Dimensional Similarity Search with Space-Filling Curves," *Proc. of Int. Conf. on Data Eng.*, pp. 615–622, 2001.
- [19] X. Liu and G. Schrack, "Encoding and Decoding the Hilbert Order," *Software-Practice and Experience*, Vol. 26, No. 12, pp. 1335–1346, Dec. 1996.
- [20] M. F. Mokbel, W. G. Aref and I. Kamel, "Analysis of Multi-Dimensional Space-Filling Curves," *Geoinformatica*, Vol. 7, No. 3, pp. 179–209, Sept. 2003.

- [21] B. Moon, H. V. Jagadish, C. Faloutsos and J. H. Saltz, “Analysis of the Clustering Properties of the Hilbert Space-Filling Curve,” *IEEE Trans. on Knowledge and Data Eng.*, Vol. 13, No. 1, pp. 124–141, Jan./Feb. 2001.
- [22] A. Papadopoulos and Y. Manolopoulos, “Performance of Nearest Neighbor Queries in R-Trees,” *Proc. of the 6th Int. Conf. on Database Theory*, pp. 394–408, 1997.
- [23] N. Roussopoulos, S. Kelley and F. Vincent, “Nearest Neighbor Queries,” *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pp. 71–79, 1995.
- [24] H. Samet, “Object-based and Image-based Object Representations,” *ACM Computing Surveys*, Vol. 36, No. 2, pp. 159–217, June 2004.
- [25] B. Seeger and H. P. Kriegel, “The Buddy-Tree: An Efficient and Robust Access Method for Spatial Data Base Systems,” *Proc. of the 16th Conf. on VLDB*, pp. 590–601, 1990.
- [26] B. Seeger, P. A. Larson, and R. McFadyen, “Reading a Set of Disk Pages,” *Proc. of the 19th Conf. on VLDB*, pp. 592–603, 1993.
- [27] J. Voros, “A Strategy for Repetitive Neighbor Finding in Images Represented by Quadtrees,” *Pattern Recognition Letters*, Vol. 18, No. 10, pp. 955–962, Oct. 1997.
- [28] R. C. W. Wong, Y. Tao, A. Fu, and X. Xiao, “On Efficient Spatial Matching,” *Proc. of the 33rd Int. Conf. On VLDB*, pp. 579–590, 2007.
- [29] M. L. Yiu, Y. Tao and N. Mamoulis, “The Bdual-Tree: Indexing Moving Objects by Space Filling Curves in the Dual Space,” *The VLDB Journal*, Vol. 17, No. 3, pp. 379–400, May 2008.
- [30] J. Zhang, M. Zhu, D. Papadias, Y. Tao and D. L. Lee, “Location-based Spatial Queries,” *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pp. 443–454, 2003.

- [31] J. Zhang, N. Mamoulis, D. Papadias and Y. Tao, “All-Nearest-Neighbors Queries in Spatial Databases,” *Proc. of the 16th Int. Conf. on Scientific and Statistical Database Management*, pp. 297–306, 2004.