

Quad-Splitting Algorithm for a Window Query on a Hilbert Curve

Chen-Chang Wu and Ye-In Chang

Dept. of Computer Science and Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan, Republic of China

E-mail: wucc@db.cse.nsysu.edu.tw

Abstract

Space-filling curves, particularly, Hilbert curves, have been extensively used to maintain spatial locality of multi-dimensional data in a wide variety of applications. A window query is an important query operation in spatial (image) databases. Given a Hilbert curve, a window query reports its corresponding orders without the need to decode all the points inside this window into the corresponding Hilbert orders. Given a query window of size $p \times q$ on a Hilbert curve of size $T \times T$, Chung *et al.* have proposed an algorithm for decomposing a window into the corresponding Hilbert orders, which needs $O(n \log T)$ time, where $n = \max(p, q)$. By employing the properties of Hilbert curves, we present an efficient algorithm, named as *Quad-Splitting*, for decomposing a window into the corresponding Hilbert orders on a Hilbert curve without individual sorting and merging steps. Although the proposed algorithm also takes $O(n \log T)$ time, it does not perform individual sorting and merging steps which are needed in Chung *et al.*'s algorithm. Therefore, experimental results show that the Quad-Splitting algorithm outperforms Chung *et al.*'s algorithm.

1 Introduction

A space-filling curve is a continuous path which passes through every point in a space once so giving a one-to-one correspondence between the coordinates of the points and the 1D-sequence numbers of points on the curve. The space-filling curve provides a way to linearly order the points of a grid. The goal is to preserve the locality; that is, points which are close in space should be stored close together in the linear order. Some examples of space-filling curves are the Peano curve [7], the RGB curve [14] and the Hilbert curve [3, 4, 10]. There have been found many applications in a variety of fields including image processing

and compression [9, 21], spatial query [7], wireless sensor networks [1], wireless broadcast system [26], neural networks [5], genome visualization [11], index of multi-dimensional data [20], spatiotemporal index [25] and bandwidth compression [2].

The Hilbert curve has been shown to have strong locality preserving properties; that is, it is the best space-filling curve in minimizing the number of clusters [15, 23]. So, Hilbert curve can scan the neighboring points in the image continuously. Moreover, the scanning order of Hilbert curve is named as Hilbert scan. Recently, some image compression methods based on Hilbert scan have been proposed [8, 9, 17, 18, 19]. Those methods are based on the Hilbert neighborhood property by a segmentation of the scanned one-dimensional data using the linear interpolation. For each segment, the beginning point, end point and their color component levels form a code, which is used to represent the segment.

A window query is an important query operation in the spatial (image) database. Given a Hilbert curve, the window query reports the corresponding orders without the need to decode all the points inside such a window. On the other hand, given a compressed image, the window query reports the responding codes without the need to decomposing the compressed image. That is, to decompress the compressed image will take a lot of time [8]. Given a query window of size $p \times q$ on a curve of size $T \times T$, employing the maximal blocks partition strategy, Liu and Schrack's encoding algorithm [22] and the related fast mapping formula, Chung *et al.* have proposed an algorithm for decomposing a window into the responding orders in $O(n \log T)$ time, where $n = \max(p, q)$ [8]. This algorithm has been applied to the window query on an arbitrary-size image [9]. Since Chung *et al.*'s algorithm needs four steps which include sorting and merging steps, it will take long time. By employing the properties of the Hilbert curve, this paper presents an efficient algorithm, Quad-Splitting, for decomposing a window into the responding orders on a Hilbert curve. Although the proposed algorithm also takes $O(n \log T)$ time, it does not perform individual sorting and merging steps which are needed in Chung *et al.*'s algorithm. Therefore, experimental results show that the proposed algorithm outperforms Chung *et al.*'s algorithm.

The rest of this paper is organized as follows. In Section 2, we briefly describe the Hilbert curves and the window query. In Section 3, we present an efficient algorithm for

decomposing a window into the responding orders on a Hilbert curve. In Section 4, we analyze the performance complexity of the proposed algorithm and make a comparison of performance of our algorithm with Chung *et al.*'s algorithm. Finally, we give a conclusion.

2 Related Work

In this section, we first introduce Hilbert curves and their applications. Then, we introduce the window query on the Hilbert curve.

2.1 Hilbert Curves and Their Applications

Hilbert curves can be thought of as finite, self-avoiding approximations of curves that pass through all points of a square. While the resolution of the Hilbert curves, denoted by k , is a positive integer, each side length of the Hilbert curve is 2^k long. Figure 1 shows three examples of the Hilbert curves of resolution $r = 1, 2$ and 3 ; that is, the sizes of these three examples are $2^1 \times 2^1$, $2^2 \times 2^2$ and $2^3 \times 2^3$, respectively. Each position on the curve is denoted by an integer, called the Hilbert order h . Consider that a Hilbert curve is of size $T \times T$, the Hilbert orders along the Hilbert curve form a strictly increasing sequence $\langle 0, 1, 2, 3, \dots, T^2 - 1 \rangle$. When we scan an image along with a Hilbert curve, we obtain the one-dimensional data having the neighborhood property.

[Figure 1 about here.]

Many different algorithms of the curve generation have been suggested [2, 3, 4, 12, 16, 22, 24]. Most of the algorithms for the curve generation are interpretations of Hilbert's original suggestion to continually divide a plane into four parts, each of these parts into four parts, and so on, calculating the necessary plotting points as the division proceeds [10, 12, 24]. This continual division of the plane continues until the required curve resolution is obtained. Breinholt and Schierz proposed a simple algorithm that can quickly and efficiently generate the points of the Hilbert curve by using the simplest recursive technique [3]. Kamata *et al.* proposed a computation method using lookup table [17] and applied it to compress an image [18, 19]. Chung *et al.* applied Liu and Schrack's encoding formula [22] to obtain the corresponding Hilbert orders and compress an image [8].

Assume that an image is scanned along the Hilbert curve. In most cases, the difference of color component levels between two neighboring pixels in the image is bounded by some small value. Due to the locality property, this image can be partitioned into some segments by using the interpolation method. For each segment, the beginning point, ending point and their color component levels form a code, which is used to represent the segment. The size of a code is less than that of represented segment in practice. If the original image is represented by a set of codes, it achieves the compression effect. Based on the Hilbert scan and the zero-order interpolation, Kamata *et al.* proposed two algorithms for compressing gray images [18] and color images [19], respectively. Jian proposed an algorithm [8] for compressing a gray image by using the first-order interpolation and the split point approximation. Chung *et al.* proposed an algorithm for compressing a gray image by using the recursive binary partition scheme and the S-tree data structure [9].

2.2 The Window Query on the Hilbert Curve

A window query is an important query operation in the spatial database. Given a Hilbert curve, the window query reports the responding sorted segments without the need to decode all the points inside this window. Chung *et al.* have proposed an algorithm for decomposing a window into the corresponding orders on a Hilbert curve [8]. They have extended this algorithm for the window query on an arbitrary-sized Hilbert curve [9]. Assume that a $p \times q$ query window W lays on a Hilbert curve of size $T \times T$. Chung *et al.*'s algorithm consisting of four steps is listed as follows.

Step 1 (Generating Maximal Blocks): Each maximal block is a square block and is denoted by (x, y, s) , where (x, y) is the coordinate of its lower left corner and s is the width of the maximal block. For the query window W , the set of maximal blocks, say M , corresponding to W is generated by using the linear-time algorithm proposed by Tsai *et al.* [8]. There are $O(n)$ maximal blocks in the worst case, where $n = \max(p, q)$. For example, in Figure 2, a query window $W(2, 2, 5, 3)$ on a Hilbert curve of size 8×8 is considered. Nine maximal blocks are obtained. $M = \{(2, 6, 1), (3, 6, 1), (4, 6, 1), (4, 5, 1), (4, 4, 1), (4, 3, 1), (4, 2, 1), (2, 4, 2), (2, 2, 2)\}$

[Figure 2 about here.]

Step 2 (Computing the Related Hilbert Orders): For each maximal block, both the minimal and maximal Hilbert orders are computed by using Liu and Schrack's encoding algorithm [22]. Since there are at most $O(n)$ maximal blocks, this step takes $O(n \log T)$ to determine the related Hilbert orders. Then, these determined Hilbert orders form a sequence S . For example, in Figure 2, a sequence $S = \{(24), (27), (36), (35), (32), (53), (54), (28, 31), (8, 11)\}$ is obtained.

Step 3 (Sorting): Since the total maximal blocks in M are not obtained as the order of the Hilbert scan, the sequence S needs to be sorted to be an increasing sequence. In this step, the quick sort algorithm is used and it takes $O(n \log n)$ time. The sorted sequence is denoted by S^* . For example, in Figure 2, the sorted sequence $S^* = \{(8, 11), (24), (27), (28, 31), (32), (35), (36), (53), (54)\}$ is obtained.

Step 4 (Merging): In order to reduce the number of Hilbert orders used, two consecutive Hilbert orders will be merged. That is, if the difference of the Hilbert orders of the begin point of the current block and the end point of the previous block is exactly one, those two consecutive Hilbert orders can be merged. Thus, the merged sorted sequence, denoted by S_{new} , is obtained. Since the sequence S^* needs to be traversed only once, this step takes $O(n)$ time. For example, in Figure 2, the merged sorted sequence $S_{new} = \{(8, 11), (24), (27, 32), (35, 36), (53, 54)\}$ is obtained. Therefore, this window is decomposed into 5 sorted segments.

From the above four steps, given a query window of size $p \times q$ on a Hilbert curve of size $T \times T$, Chung *et al.*'s algorithm takes $O(n \log T)$ time to perform a window query on a Hilbert curve, where $n = \max(p, q)$. However, their four steps that include sorting and merging steps take long time.

3 The Proposed Algorithm

In this section, we first introduce some properties of Hilbert curves. Then, we present an algorithm, Quad-Splitting, for decomposing a window into sorted segments on a Hilbert curve.

3.1 Observations

Breinholt and Schierz proposed a simple algorithm that can quickly and efficiently generate the points of the Hilbert curve by using the simplest recursive technique [3]. This algorithm deconstructs the Hilbert curve into a set of unit shapes. In addition, the relative position and rotation of each unit shape is defined by its sequential position in the curve generation. In order to let the points of the curve be plotted in the correct position and the correct order, this algorithm defines four possible orientations of the unit shape to describe both the rotation of the unit shape and its start and end points. That is, the Hilbert curves start at one corner of a square and end at one of the two neighboring corners of the square. Figure 3 illustrates four of the possible orientations of the unit shape. For *Orientation A* unit shape, the lower left corner is the start point and the lower right corner is the end point.

[Figure 3 about here.]

Suppose a Hilbert curve is evenly partitioned into four parts. Suppose the lower left part, the lower right part, the upper left part and the upper right part are P_0 , P_1 , P_2 and P_3 , respectively. We have the following three observations.

Observation 1: For each part, it is still a Hilbert curve and its orientation can be determined according to the orientation of the original Hilbert curve.

For example, Figure 4 shows that a $2^3 \times 2^3$ Hilbert curve of *Orientation A* is obtained. After it is evenly partitioned into four parts, we find that each part is also a $2^2 \times 2^2$ Hilbert curve. Suppose the orientations of the parts P_0 , P_1 , P_2 and P_3 are O_0 , O_1 , O_2 and O_3 , respectively, as shown in Figure 5. For Figure 4, we have $O_0 = B$, $O_1 = D$, $O_2 = A$ and $O_3 = A$.

[Figure 4 about here.]

[Figure 5 about here.]

Therefore, Figure 6 shows four cases that the orientations of parts are determined after a curve is evenly partitioned into four parts for four orientations. For any one of *Orientation A* Hilbert curves as shown in Figure 6-(a), after they are partitioned into four parts, each

part is also a Hilbert curve and the orientations of these parts are listed as follows: $O_0 = B$, $O_1 = D$, $O_2 = A$ and $O_3 = A$. Similarly, the orientations of parts for *Orientations B, C and D* as shown in Figures 6-(b), 6-(c) and 6-(d), respectively.

[Figure 6 about here.]

Observation 2: For each part, since its associated Hilbert orders form a strictly increasing sequence, the minimal Hilbert order of each part can be determined.

For example, in Figure 4, there is a $2^3 \times 2^3$ Hilbert curve of *Orientation A* and its Hilbert orders start from 0 to 63 ($= 0 + 2^3 * 2^3 - 1$). That is, suppose the curve is of size $2^r \times 2^r$ and its minimal Hilbert order is denoted by *min_order*, we can get that the maximal Hilbert order is $(min_order + 2^r * 2^r - 1)$. After this $2^3 \times 2^3$ curve is evenly partitioned into four parts, each part is a $2^2 \times 2^2$ Hilbert curve. We can find that the Hilbert orders of the lower left part form a strictly increasing sequence $\langle 0, 1, 2, \dots, 15 \rangle$; that is, the minimal Hilbert order is 0 and the maximal Hilbert order is 15 ($= 0 + 2^2 * 2^2 - 1$). For the upper left part, its Hilbert orders also form a strictly increasing sequence $\langle 16, 17, 18, \dots, 31 \rangle$; that is, the minimal Hilbert order is 16 and the maximal Hilbert order is 31 ($= 16 + 2^2 * 2^2 - 1$). For the upper right part, its Hilbert orders also form a strictly increasing sequence $\langle 32, 33, 34, \dots, 47 \rangle$; that is, the minimal Hilbert order is 32 and the maximal Hilbert order is 47 ($= 32 + 2^2 * 2^2 - 1$). For the lower right part, its Hilbert orders also form a strictly increasing sequence $\langle 48, 49, 50, \dots, 63 \rangle$, that is, the minimal Hilbert order is 48 and the maximal Hilbert order is 63 ($= 48 + 2^2 * 2^2 - 1$).

Therefore, suppose the minimal Hilbert orders of the lower left part, the lower right part, the upper left part and the upper right part are min_0 , min_1 , min_2 and min_3 , respectively. For any one of $2^r \times 2^r$ Hilbert curves of *Orientation A* whose minimal Hilbert order is denoted by *min_order*, after it is evenly partitioned into four parts, the minimal Hilbert orders of each part are $min_0 = min_order$, $min_1 = min_order + 2^{r-1} * 2^{r-1} * 3$, $min_2 = min_order + 2^{r-1} * 2^{r-1}$ and $min_3 = min_order + 2^{r-1} * 2^{r-1} * 2$ as shown in Figure 7-(a). Similarly, for any one of $2^r \times 2^r$ Hilbert curves of *Orientation B*, *Orientation C* and *Orientation D*, the minimal Hilbert orders of their parts are determined as shown in Figure 7-(b), Figure 7-(c) and Figure 7-(d), respectively.

[Figure 7 about here.]

Observation 3: For any kind of window queries, there are nine relationships among a Hilbert curve.

In [6], Chang *et al.* have proposed that when an object is lying on the space, only nine cases are possible based on decomposing the whole region into four regions. Hence, after a Hilbert curve is evenly partitioned into four parts, when a query window is lying on this curve, we also find that only nine relationships are possible as shown in Figure 8. Thus, for a Hilbert curve of size $T \times T$ and the query window $W(x, y, p, q)$, where the size of this window is $p \times q$ and (x, y) is the coordinate of the lower left corner of this window. Suppose there is a query window of size $p \times q$ and the coordinates of its lower left corner and upper right corner are llw and urw , respectively. We have $llw = (x, y)$ and $urw = (x + q, y + p)$. The possible nine relationships are described as followed:

[Figure 8 about here.]

- Type 0: When $llw \in P_0$ and $urw \in P_3$ as shown in Figure 8-(a), the condition $(x < T/2, y < T/2, (x + q) \geq T/2$ and $(y + p) \geq T/2)$ is true, we can find that this window can also be partitioned into four sub-windows simultaneously.
- Type 1: When $llw \in P_2$ and $urw \in P_3$ as shown in Figure 8-(b), the condition $(x < T/2, y \geq T/2, (x + q) \geq T/2$ and $(y + p) \geq T/2)$ is true, we can find that this window can also be partitioned into two sub-windows simultaneously.
- Type 2: When $llw \in P_0$ and $urw \in P_1$ as shown in Figure 8-(c), the condition $(x < T/2, y < T/2, (x + q) \geq T/2$ and $(y + p) < T/2)$ is true, we can find that this window can also be partitioned into two sub-windows simultaneously.
- Type 3: When $llw \in P_0$ and $urw \in P_2$ as shown in Figure 8-(d), the condition $(x < T/2, y < T/2, (x + q) < T/2$ and $(y + p) \geq T/2)$ is true, we can find that this window can also be partitioned into two sub-windows simultaneously.
- Type 4: When $llw \in P_1$ and $urw \in P_3$ as shown in Figure 8-(e), the condition $(x \geq T/2, y < T/2, (x + q) \geq T/2$ and $(y + p) \geq T/2)$ is true, we can find that this window can also be partitioned into two sub-windows simultaneously.

- Type 5: When both llw and $urw \in P_2$ as shown in Figure 8-(f), the condition $(x < T/2, y \geq T/2, (x + q) < T/2$ and $(y + p) \geq T/2)$ is true, we can find that this window is only in the upper left part.
- Type 6: When both llw and $urw \in P_3$ as shown in Figure 8-(g), the condition $(x \geq T/2, y \geq T/2, (x + q) \geq T/2$ and $(y + p) \geq T/2)$ is true, we can find that this window is only in the upper right part.
- Type 7: When both llw and $urw \in P_0$ as shown in Figure 8-(h), the condition $(x < T/2, y < T/2, (x + q) < T/2$ and $(y + p) < T/2)$ is true, we can find that this window is only in the lower left part.
- Type 8: When both llw and $urw \in P_1$ as shown in Figure 8-(i), the condition $(x \geq T/2, y < T/2, (x + q) \geq T/2$ and $(y + p) < T/2)$ is true, we can find that this window is only in the lower right part.

3.2 The Quad-Splitting Approach

Assume that there is a query window W of size $p \times q$ inside an *Orientation* $oABCD$ of size $T \times T$ Hilbert curve whose minimal Hilbert order is denoted as $mino$, which has its lower left corner at position (x, y) . Such a Hilbert curve will be denoted as $HC(oABCD, T, mino)$ and a query window as $W(x, y, p, q)$. We can obtain a sorted sequence $S = \{s_0, s_1, \dots, s_k\}$, where s_k is called a segment that uses both the minimal and maximal Hilbert orders, $(mins_k, maxs_k)$, to identify the consecutive Hilbert orders of a segment. Each segment does not intersect the other segment. Moreover, if $i > j$, then we have $mins_i > mins_j$. For example in Figure 2, a query window $W(2, 2, 5, 3)$ in a Hilbert curve $HC(A, 8, 0)$ is considered. We can obtain a sorted sequence $S = \{(8, 11), (24, 24), (27, 32), (35, 36), (53, 54)\}$.

We now present an efficient algorithm shown in Figure 9 to decomposing a window into sorted segments on a Hilbert curve. We can call procedure $QSplit(oABCD, T, mino, x, y, p, q)$ to decompose the window $W(x, y, p, q)$ into sorted segments on the Hilbert curve $HC(oABCD, T, mino)$. This algorithm is implemented in a recursive procedure that modifies its calling parameters as it converges a sorted sequence on the curve. For example, in Figure 2, to obtain a sorted sequence, the corresponding orders, the first call is procedure

$QSplit(A, 8, 0, 2, 2, 5, 3)$.

[Figure 9 about here.]

In procedure $QSplit(oABCD, T, mino, x, y, p, q)$, first, if the size of the query window is equal to the size of the curve, the result of this window is the same as the orders of a Hilbert curve that its size is $T \times T$ and the minimal Hilbert order is $mino$. Based on *Observation 1*, we will obtain the minimal Hilbert order, $mins = mino$, and the maximal Hilbert order, $maxs = mino + T * T - 1$, in this window. Then, the Hilbert segment $(mins, maxs)$ is added to a sequence S by calling procedure $addSequence$ which will merge the two consecutive Hilbert segments as shown in Figure 10.

[Figure 10 about here.]

Otherwise, if the size of a window is not equal to the size of a curve, we will partition this curve into four parts evenly. Assume that the lower left part, the lower right part, the upper left part and the upper right part are denoted as SHC_0 , SHC_1 , SHC_2 and SHC_3 , respectively. Based on *Observation 1*, the orientation of each part can be determined. For example, there are four parts, $SHC_0 = HC(B, 4, 0)$, $SHC_1 = HC(D, 4, 48)$, $SHC_2 = HC(A, 4, 16)$ and $SHC_3 = HC(A, 4, 32)$ in Figure 11.

[Figure 11 about here.]

Moreover, for each part, the overlapping between the window and this part produces the sub-window. Assume that the sub-windows in the lower left part, the lower right part, the upper left part and the upper right part are denoted as SW_0 , SW_1 , SW_2 and SW_3 , respectively. In Figure 12-(a), for a window $W(x, y, p, q)$ lying on all four parts, *i.e.* Type 0, we have $SW_0 = W(x, y, N/2 - y, N/2 - x)$, $SW_1 = W(0, y, N/2 - y, x + q - N/2)$, $SW_2 = W(x, 0, y + p - N/2, N/2 - x)$ and $SW_3 = W(0, 0, y + p - N/2, x + q - N/2)$. For the other types, the sub-windows are obtained in Figure 12. For example in Figure 11, there are four sub-windows, $SW_0 = W(2, 2, 2, 2)$, $SW_1 = W(0, 2, 2, 1)$, $SW_2 = W(2, 0, 3, 2)$ and $SW_3 = W(0, 0, 3, 1)$ on the parts $SHC_0 = HC(B, 4, 0)$, $SHC_1 = HC(D, 4, 48)$, $SHC_2 = HC(A, 4, 16)$ and $SHC_3 = HC(A, 4, 32)$, respectively.

[Figure 12 about here.]

Next, we can perform the query for each sub-window to obtain the responding Hilbert orders recursively. But, this result may not be sorted segments, if we do not care the order of these sub-window queries. However, according to *Observations* 1 and 2, we know that each part is a Hilbert curve and the minimal Hilbert order of each part can be determined based on the orientation of this original curve. So, in order to obtain sorted segments in one step, the order of each sub-window query called must be determined based on the orientation of the original curve. Following previous observations, we can find that there are 36 cases for a window laying among four partitioned parts as shown in Figure 13. We can determine which sub-windows will be queried and the order of sub-windows queried based on these 36 cases. For example, when a query window on the Hilbert curve of *Orientation A* is of *type 1* relationship, we can find that there are two sub-windows, SW_2 and SW_3 , performed in the specific order that the sub-window SW_2 is performed first and SW_3 is performed next. For another example, when a query window on the Hilbert curve of *Orientation A* is of *type 4* relationship, we can find that there are two sub-windows, SW_1 and SW_3 , performed in the specific order that the sub-window SW_3 is performed first and SW_1 is performed next.

[Figure 13 about here.]

Hence, in order to simplify the *QSplit* procedure, these 36 cases are classified by the orientation of the original curve. That is, there are four sub-procedures, *QSplitA* (as shown in Figure 14), *QSplitB* (as shown in Figure 15), *QSplitC* (as shown in Figure 16) and *QSplitD* (as shown in Figure 17), performed for the *Orientations A, B, C* and *D* curves, respectively. For each sub-procedure, we can determine which type ($0 \cdots 8$) the window holds by calling function *determineType* that can determine which sequence its all sub-windows will be performed in. Therefore, we can perform each sub-window query to obtain the responding sorted segments in the determined order of sub-window recursively.

[Figure 14 about here.]

[Figure 15 about here.]

[Figure 16 about here.]

[Figure 17 about here.]

For example in Figure 11, there are four sub-windows, SW_0 , SW_2 , SW_3 and SW_1 , performed in sequence. That is, since the curve $HC(A, 8, 0)$ has *Orientation A*, we will call procedure $QSplit(B, 4, 0, 2, 2, 2, 2)$, $QSplit(A, 4, 16, 2, 0, 3, 2)$, $QSplit(A, 4, 32, 0, 0, 3, 1)$ and $QSplit(D, 4, 48, 0, 2, 2, 1)$ in sequence. Next, for the procedure $QSplit(B, 4, 0, 2, 2, 2, 2)$, since the size of the window $W(2, 2, 2, 2)$ is not that of the Hilbert curve $HC(B, 4, 0)$ and this window $W(2, 2, 2, 2)$ on the curve $HC(B, 4, 0)$ is of *type 6*, we can call procedure $QSplit(B, 2, 8, 0, 0, 2, 2)$ to decompose the window $W(0, 0, 2, 2)$ on the curve $HC(B, 2, 8)$ into the corresponding sorted segments. Then, for procedure $QSplit(B, 2, 8, 0, 0, 2, 2)$, since the size of window is equal to that of curve, it produces the order segment $(8, 11)$ and adds this segment into the sorted sequence by calling procedure *addSequence*. This algorithm works by recursively calling itself with modified its parameters with each call, until the size of the window is equal to the size of the curve. Figure 18 shows the process of executing the procedure $QSplit(A, 8, 0, 2, 2, 5, 3)$ and Table 1 shows the detail of these nodes. First, the procedure $QSplit(A, 8, 0, 2, 2, 5, 3)$, denoted as node 0, will call procedures $QSplit(B, 4, 0, 2, 2, 2, 2)$ (node 1), $QSplit(A, 4, 16, 2, 0, 3, 2)$ (node 2), $QSplit(A, 4, 32, 0, 0, 3, 1)$ (node 3) and $QSplit(D, 4, 48, 0, 2, 2, 1)$ (node 4) in sequence. Next, the procedure $QSplit(B, 4, 0, 2, 2, 2, 2)$ will call procedure $QSplit(B, 2, 8, 0, 0, 2, 2)$ (node 5) and obtain the order segment $(8, 11)$ and add this segment into the sorted sequence. Finally, a sorted sequence S is obtained. $S = \{(8,11), (24), (27,32), (35,36), (53,54)\}$

[Figure 18 about here.]

[Table 1 about here.]

4 Performance

In this section, we first analyze the time complexity of the proposed algorithm. Then, we compare the performance of the proposed algorithm with that of Chung *et al.*'s algorithm.

4.1 Analysis of Time Complexity

From procedure *QSplit*, it is known that the sorted Hilbert orders are obtained, while the size of the window (sub-window) is equal to the size of the curve (sub-curve). If the size of the window (sub-window) is not equal to the size of the curve (sub-curve), we use procedures *QSplitA*, *QSplitB*, *QSplitC* and *QSplitD* to evenly partition this curve (sub-curve) into four parts (sub-curves). Since the quadtree is based on a regular decomposition of space into maximal blocks [13] whose sides are of size a power of two, and are placed at predetermined positions, the decomposition can be represented as a tree of out-degree 4, with the root (at level 0) corresponding to the whole image, and each level- d node corresponding to a block of side length $T/2^d$. This decomposition is equivalent to that procedure *QSplit* evenly partitions a curve into four parts recursively. Then, we have the following result.

Lemma 1. For any performed procedure *QSplit*, if the size of the window (sub-window) is equal to the size of the curve (sub-curve), this window (sub-window) is a maximal block.

For example, in Figure 19, there are nine maximal blocks for a query window $W(2, 2, 5, 3)$ in an 8×8 Hilbert curve. We find the result that the leaves in Figure 18 are equivalent to the maximal blocks in Figure 19, since their lower left coordinates and size are the same. For example, the leaf, node 5 in Figure 18, is equivalent to the maximal block, B_8 in Figure 19.

[Figure 19 about here.]

If we partition the query window into some maximal blocks and use each maximal block as a query sub-window to perform procedure *QSplit*, we find that the execution time of the proposed algorithm is dominated by the total number of generated maximal blocks. That is, the execution time of the proposed algorithm is bounded by the total execution time for each query sub-window. Then, we have the following result.

Lemma 2. Given a query window and its maximal blocks are obtained, the execution time of the proposed algorithm performed for a query window is bounded by the total execution time of the proposed algorithm for each maximal blocks.

For example, in Figure 19, the execution time of *QSplit*(8, 0, 2, 2, 5, 3) is bounded by the

total execution time of each query sub-window which includes maximal blocks B_0, B_1, \dots, B_8 . Given a curve of size $T \times T$, if the query window is the maximal block size of $T \times T$, procedure *QSplit* is performed 1 time. If the query window is the maximal block of size $T/2 \times T/2$, procedure *QSplit* is performed 2 times. However, If the query window is the maximal block of size $T/2^k \times T/2^k$, procedure *QSplit* is performed k times. Suppose the maximal block is of size $2^m \times 2^m (= T/2^{(\log_2 T - m)} \times T/2^{(\log_2 T - m)})$. It is known that procedure *QSplit* is performed $(\log_2 T - m)$ times to decompose this maximal block into the corresponding Hilbert orders. Then, we have the following result.

Lemma 3. Given a curve of size $T \times T$, if the query window is the maximal block of size $2^m \times 2^m$, procedure *QSplit* is performed $(\log_2 T - m)$ times to decompose the window (maximal block) into the corresponding Hilbert orders.

For example in Figure 18, we find that procedure *QSplit* has been performed 3 and 4 times in node 5 and 11, respectively.

Lemma 4. Given a query window of size $p \times q$, there are $O(n)$ maximal blocks in the worst case, where $n = \max(p, q)$ [8].

From Lemma 3, it is known that the maximal block of size 1×1 takes the longest time, $\log_2 T$, among all maximal blocks. From Lemma 2 and Lemma 4, when all the maximal blocks are of size 1×1 is the worst case of the proposed algorithm. Therefore, the proposed algorithm takes $O(n \log T)$ time to decompose the window into the corresponding Hilbert orders, where $n = \max(p, q)$.

Theorem 1. Given a query window of size $p \times q$, the proposed algorithm takes $O(n \log T)$ time to decompose the window (maximal block) into the corresponding Hilbert orders, where $n = \max(p, q)$.

Although the proposed algorithm also takes $O(n \log T)$ time, it does not perform individual sorting and merging steps which are needed in Chung *et al.*'s algorithm. In the next sub-section, the experiment results show that the proposed algorithm improves Chung *et al.*'s algorithm significantly in terms of the processing time.

4.2 Experiment Results

In this subsection, we compare the performance of the Quad-Splitting algorithm with that of Chung *et al.*'s algorithm [8] and evaluate the computing time for the Quad-Splitting algorithm. All experiments have been performed on the mobile AMD Athlon XP micro-processor with 1.33G MHz and 1G RAM. The operating system is MS-Windows XP and all programs are developed by Java 1.6.0. Assume that these inputs are a $T \times T$ Hilbert curve and the size of a query window is $p \times q$. We use the processing time to generate the corresponding sorted segments in the query window as the performance measure.

First, we set $p = q = n$, $T = 1024$ and randomly choose the starting points to generate 10000 query windows. The experimental results for square windows are shown in Table 2 and Figure 20. For square windows with different widths, the total execution time for performing these 10000 window queries using the Quad-Splitting algorithm and Chung *et al.*'s algorithm, which are denoted by T_{ours} and T_{Chung} , respectively. The improvement ratio of the execution time required in the proposed algorithm over the Chung *et al.*'s algorithm is denoted by $R = (T_{Chung} - T_{ours})/T_{Chung} \times 100\%$ as shown in the final column of Table 2. From Table 2 and Figure 20, it is observed that the Quad-Splitting algorithm outperforms Chung *et al.*'s algorithm and has about 92-97% time improvement. From Figure 20, the larger the query window becomes, the better the performance is. The reason is that as the number of segments increases, the processing time of the sorting and merging steps in Chung *et al.*'s algorithm will increase.

[Table 2 about here.]

[Figure 20 about here.]

Second, the arbitrary rectangular windows are used as inputs. Ten types of area are used in the experimentations and they are 1000, 2000, 3000, \dots , and 10000. For each specific area, 10000 query windows are generated by randomly choosing the starting points, the width and the height of the window. The comparison of the execution time between the Quad-Splitting algorithm and Chung *et al.*'s algorithm are shown in Table 3 and Figure 21. From Table 3 and Figure 21, it is observed that the Quad-Splitting algorithm outperforms Chung *et al.*'s algorithm and has about 95-98% time improvement. Since Chung *et al.*'s

algorithm must execute the sorting and the merging steps, their execution time is more than ours. From Figure 21, the larger the query window becomes, the better the performance is. The reason is that as the number of segments increases, the processing time of the sorting and merging steps in Chung *et al.*'s algorithm will increase.

[Table 3 about here.]

[Figure 21 about here.]

Next, we study the effect of the size of the Hilbert curve on the query performance for the Quad-Splitting algorithm. Assume that these inputs are an $T \times T$ Hilbert curve and the size of a query window is 10×10 . Seven types of the Hilbert curve are used in the experimentations and their side sizes are 16, 16^2 , 16^3 , \dots , and 16^7 . For each specific curve, 100000 query windows are generated by randomly choosing the starting points. The experimental results are shown in Table 4 and Figure 22. From Figure 22, it is observed that the execution time of the Quad-Splitting algorithm is approximately linear by varying the base-16 logarithm of the side size of the curve, *i.e.* $\log_{16}T$. Therefore, given a fixed size window and a Hilbert curve of size $T \times T$, the Quad-Splitting algorithm takes $O(\log T)$ time to decompose the window into sorted segments on the Hilbert curve. This fits Theorem 1.

[Table 4 about here.]

[Figure 22 about here.]

Finally, we evaluate the effect of the size of the window on the query performance for the Quad-Splitting algorithm. Assume that these inputs are an 1024×1024 Hilbert curve and the size of a query window is $p \times q$. Ten types of the square window ($p = q$) are used in the experimentations and their heights (widths) are 100, 200, 300, \dots , and 1000. For each specific window, 1000 query windows are generated by randomly choosing the starting points. From Figure 23, it is observed that the execution time of the Quad-Splitting algorithm is nearly linear by varying the side size of the window. Therefore, given a window of size $p \times q$ on a fixed size Hilbert curve, the Quad-Splitting algorithm takes $O(n)$ time to decompose the window into sorted segments on the Hilbert curve, where $n = \max(p, q)$. This fits Theorem 1.

[Figure 23 about here.]

5 Conclusion

In this paper, we have presented the Quad-Splitting algorithm for decomposing a window query into sorted segments on a Hilbert curve based on the properties of Hilbert curve. Given a query window of size $p \times q$ on a Hilbert curve of size $T \times T$, the Quad-Splitting algorithm takes $O(n \log T)$ time to perform the window query, where $n = \max(p, q)$. Although the proposed algorithm also takes $O(n \log T)$ time, it does not perform individual sorting and merging steps which are needed in Chung *et al.*'s algorithm. From the simulation results, we have shown that the Quad-Splitting algorithm improves Chung *et al.*'s algorithm up to 98% in terms of the processing time. The experimental results also confirm the time complexity analysis.

6 Acknowledgments

This research was supported in part by the National Science Council of Republic of China under Grant No. NSC-94-2213-E-110-003 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.

References

- [1] Bahi, J. M., Makhoul, A., and Mostefaoui, A.: 'Localization and Coverage for High Density Sensor Networks', Computer Communications, 2008, 31, (4), pp. 770–781
- [2] Bially, T.: 'A Space-Filling Curves: Their Generation and Their Application to Bandwidth Reduction', IEEE Trans. on Information Theory, 1969, IT-15, (6), pp. 658-664
- [3] Breinholt, G., and Schierz, C.: 'Algorithm 781: Generating Hilbert's Space-Filling Curve by Recursion', ACM Trans. on Mathematical Software, 1998, 24, (2), pp. 184-189
- [4] Butz, A. R.: 'Alternative Algorithm for Hilbert's Space-Filling Curve', IEEE Trans. on Computers, 1971, C-20, (4), pp. 424-426

- [5] Castro, J., Georgiopoulos, M., Demara, R., and Gonzalez, A.: 'Data-Partitioning Using the Hilbert Space Filling Curves: Effect on the Speed of Convergence of Fuzzy ARTMAP for Large Database Problems', *Neural Networks*, 2005, 18, (7), pp. 967-984
- [6] Chang, Y. I., Liao, C. H., and Chen, H. L.: 'NA-Trees: A Dynamic Index for Spatial Data', *Journal of Information Science and Engineering*, 2003, 19, (1), pp. 103-139
- [7] Chen, H. L., and Chang, Y. I.: 'Neighbor-Finding Based on Space-Filling Curves', *Information Systems*, 2005, 30, (3), pp. 205-226
- [8] Chung, K. L., Tasi, Y. H., and Hu, F. C.: 'Space-Filling Approach for Fast Window Query on Compressed Images', *IEEE Trans. on Image Processing*, 2000, 9, (12), pp. 2109-2116
- [9] Chung, K. L., Huang, Y. L., and Liu, Y. W.: 'Efficient Algorithms for Coding Hilbert Curve of Arbitrary-sized Image and Application to Window Query', *Information Sciences*, 2007, 177, (10), pp. 2130-2151
- [10] Cole, A. J.: 'A Note on Space Filling Curves', *Software: Practice and Experience*, 1983, 13, (12), pp. 1181-1189
- [11] Deng, X., Deng, X., Rayner, S., Liu, X., Zhang, Q., Yang, Y., and Li, N.: 'DHPC: A New Tool to Express Genome Structural Features', *Genomics*, 2008, 91, (5), pp. 476-483
- [12] Goldschlager, L. M.: 'Short Algorithms for Space-Filling Curves', *Software: Practice and Experience*, 1981, 11, (1), pp. 99-100
- [13] Guido, P.: 'An optimal algorithm for decomposing a window into maximal quadtree blocks', *Acta Informatica*, 1999, 36, (4), pp. 257-266
- [14] Jagadish, H. V.: 'Linear Clustering of Objects with Multiple Attributes', *Proc. of ACM SIGMOD Conf.*, 1990, pp. 332-342
- [15] Jagadish, H. V.: 'Analysis of the Hilbert Curve for Representing Two-Dimensional Space', *Information Processing Letters*, 1997, 62, (1), pp. 17-22

- [16] Jin, G., and Mellor-Crummey, J.: 'SFCGen: A Framework for Efficient Generation of Multi-Dimensional Space-Filling Curves by Recursion', *ACM Trans. on Mathematical Software*, 2005, 31, (1), pp. 120-148
- [17] Kamata, S., Eaxon, R. O., and Kawaguchi, E.: 'An Implementation of the Hilbert Scanning Algorithm and its Application to Data Compression', *IEICE Trans. Information and Systems*, 1993, E76-D, (4), pp. 420-428
- [18] Kamata, S., Niimi, M., and Kawaguchi, E.: 'A Gray Image Compression Using a Hilbert Scan', *Proc. of IEEE Int. Conf. on Pattern Recognition*, Aug. 1996, pp. 905-909
- [19] Kamata, S., Niimi, M., and Bandoh, Y.: 'Color Image Compression Using a Hilbert Scan', *Proc. of IEEE Int. Conf. on Pattern Recognition*, Aug. 1998, pp. 1575-1578
- [20] Lawder, J. K., and King, P. J. H.: 'Querying Multi-Dimensional Data Indexed Using the Hilbert Space-Filling Curve', *ACM SIGMOD Record*, 2001, 30, (1), pp. 19-24
- [21] Liang, J. Y., Chen, C. S., Huang, C. H., and Liu, L.: 'Lossless Compression of Medical Images Using Hilbert Space-Filling Curves', *Computerized Medical Imaging and Graphics*, 32, (3), pp. 174-182
- [22] Liu, X., and Schrack, G. F.: 'Encoding and Decoding the Hilbert Order', *Software: Practice and Experience*, 1996, 26, (12), pp. 1335-1346
- [23] Moon, B., Jagadish, H. V., Faloutsos, C., and Saltz, J. H.: 'Analysis of the Clustering Properties of the Hilbert Space-Filling Curve', *IEEE Trans. on Knowledge and Data Engineering*, 2001, 13, (1), pp. 124-141
- [24] Witten, I. H., and Wyvill, B.: 'On the Generation and Use of Space-filling Curves', *Software: Practice and Experience*, 1983, 13, (6), pp. 519-525
- [25] Yiu, M. L., Tao, Y., and Mamoulis, N.: 'The Bdual-Tree: Indexing Moving Objects by Space Filling Curves in the Dual Space', *The VLDB Journal*, 2008, 17, (3), pp. 379-400

- [26] Zheng,B., Lee, W. C., and Lee, D. L.: 'Spatial Queries in Wireless Broadcast Systems',
Wireless Networks, 2004, 10, (6), pp. 723-736

List of Figures

1	Hilbert curves for resolutions $r = 1, 2$ and 3	22
2	Sample 1: A query window of size 5×3 in a Hilbert curve of size 8×8 . .	23
3	Four orientations of the unit shape: (a) <i>Orientation A</i> ; (b) <i>Orientation B</i> ; (c) <i>Orientation C</i> ; (d) <i>Orientation D</i>	24
4	A Hilbert curve of size $2^3 \times 2^3$	25
5	The orientation of each part after a curve is partitioned into four parts . .	26
6	The orientation of each part after a Hilbert curve is evenly partitioned into four parts for different orientations: (a) <i>Orientation A</i> ; (b) <i>Orientation B</i> ; (c) <i>Orientation C</i> ; (d) <i>Orientation D</i>	27
7	The minimal orders of each part after partitioning each of different orienta- tion Hilbert curve into four parts: (a) <i>Orientation A</i> ; (b) <i>Orientation B</i> ; (c) <i>Orientation C</i> ; (d) <i>Orientation D</i>	28
8	9 possible relationships for any kind of window queries: (a) Type 0; (b) Type 1; (c) Type 2; (d) Type 3; (e) Type 4; (f) Type 5; (g) Type 6; (h) Type 7; (i) Type 8.	29
9	Procedure <i>QSplit</i>	30
10	Procedure <i>addSequence</i>	31
11	Four sub-window queries, SW_0 , SW_1 , SW_2 and SW_3 among four parts after a Hilbert curve is evenly partitioned into four parts.	32
12	The sub-windows after partitioning the window for nine types: (a) Type 0; (b) Type 1; (c) Type 2; (d) Type 3; (e) Type 4; (f) Type 5; (g) Type 6; (h) Type 7; (i) Type 8.	33
13	36 Cases	34
14	Procedure <i>QSplitA</i>	35
15	Procedure <i>QSplitB</i>	36
16	Procedure <i>QSplitC</i>	37
17	Procedure <i>QSplitD</i>	38
18	Trace of Sample 1	39
19	The maximal blocks of the query window for Sample 1	40
20	A comparison of the total execution time between the Quad-Aplitting algo- rithm and Chung <i>et al.</i> 's algorithm by varying the side length of the square window	41
21	A comparison of the total execution time between the Quad-Splitting algo- rithm and Chung <i>et al.</i> 's algorithm by varying the size of the arbitrary rectangular window	42
22	The total execution time of the Quad-Splitting algorithm for 100000 10×10 window queries by varying the side size of the Hilbert curve	43
23	The total execution time of the Quad-Splitting algorithm for 1000 window queries on a 1024×1024 Hilbert curve by varying the size of the square window	44

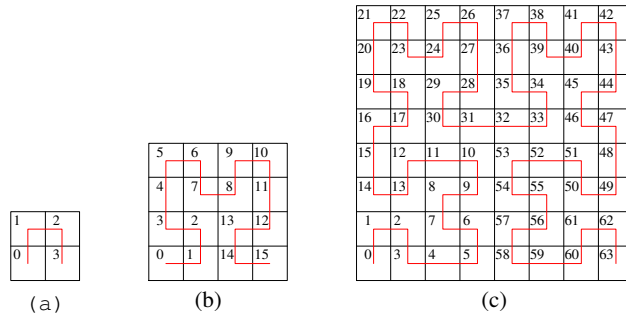


Figure 1: Hilbert curves for resolutions $r = 1, 2$ and 3

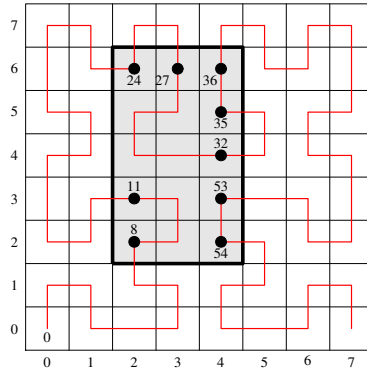
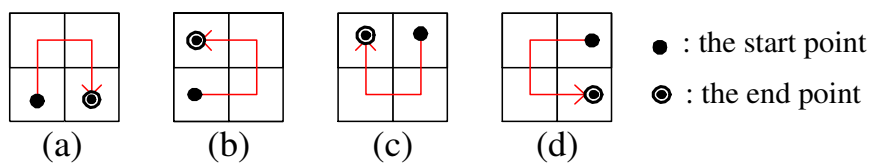


Figure 2: Sample 1: A query window of size 5×3 in a Hilbert curve of size 8×8



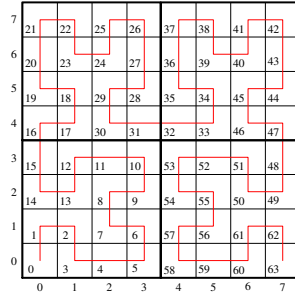


Figure 4: A Hilbert curve of size $2^3 \times 2^3$

\mathcal{O}_2	\mathcal{O}_3
\mathcal{O}_0	\mathcal{O}_1

Figure 5: The orientation of each part after a curve is partitioned into four parts

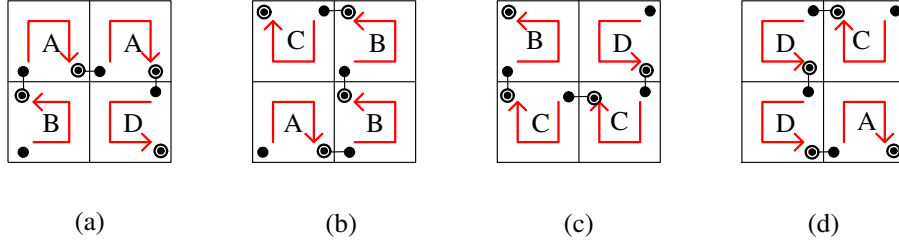


Figure 6: The orientation of each part after a Hilbert curve is evenly partitioned into four parts for different orientations: (a) *Orientation A*; (b) *Orientation B*; (c) *Orientation C*; (d) *Orientation D*.

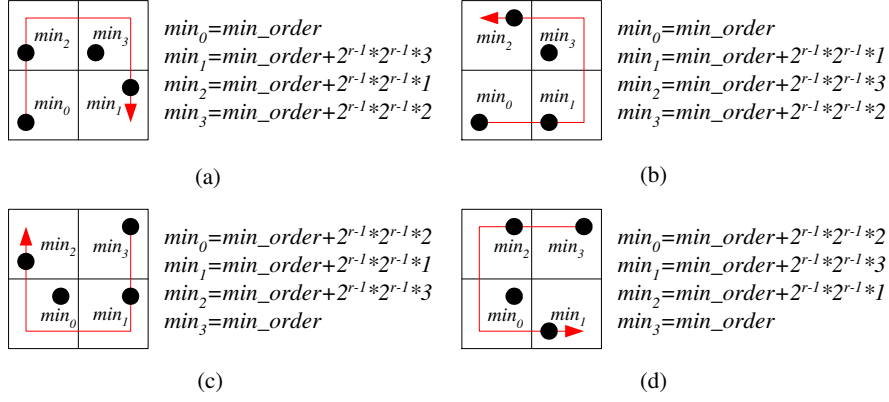


Figure 7: The minimal orders of each part after partitioning each of different orientation Hilbert curve into four parts: (a) *Orientation A*; (b) *Orientation B*; (c) *Orientation C*; (d) *Orientation D*.

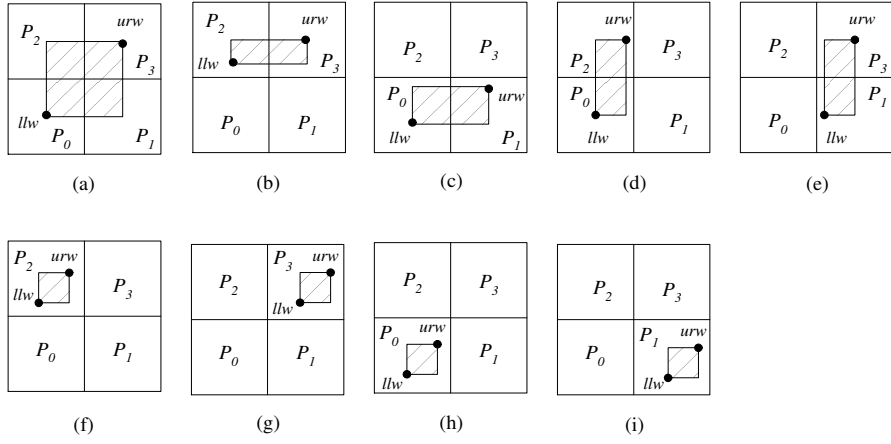


Figure 8: 9 possible relationships for any kind of window queries: (a) Type 0; (b) Type 1; (c) Type 2; (d) Type 3; (e) Type 4; (f) Type 5; (g) Type 6; (h) Type 7; (i) Type 8.

```

Procedure QSplit (oABCD, T, mino, x, y, p, q);
begin
  if (T = p) and (T = q) then addSequence(mino, mino + T * T - 1)
  else
    begin
      case oABCD of
        A : QSplitA(T, mino, x, y, p, q);
        B : QSplitB(T, mino, x, y, p, q);
        C : QSplitC(T, mino, x, y, p, q);
        D : QSplitD(T, mino, x, y, p, q);
      end;
    end;
  end;

```

Figure 9: Procedure *QSplit*

```

Procedure addSequence (mins, maxs);
begin
  if a sequence S is null
  then append this segment (mins, maxs) to S
  else
  begin
    if the last segment in S and this segment (mins, maxs) are consecutive
    then merge these two segments
    else
      append this segment (mins, maxs) to S
    end;
  end;

```

Figure 10: Procedure *addSequence*

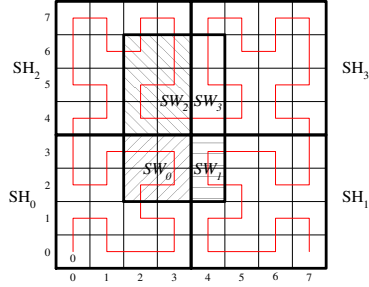


Figure 11: Four sub-window queries, SW_0 , SW_1 , SW_2 and SW_3 among four parts after a Hilbert curve is evenly partitioned into four parts.

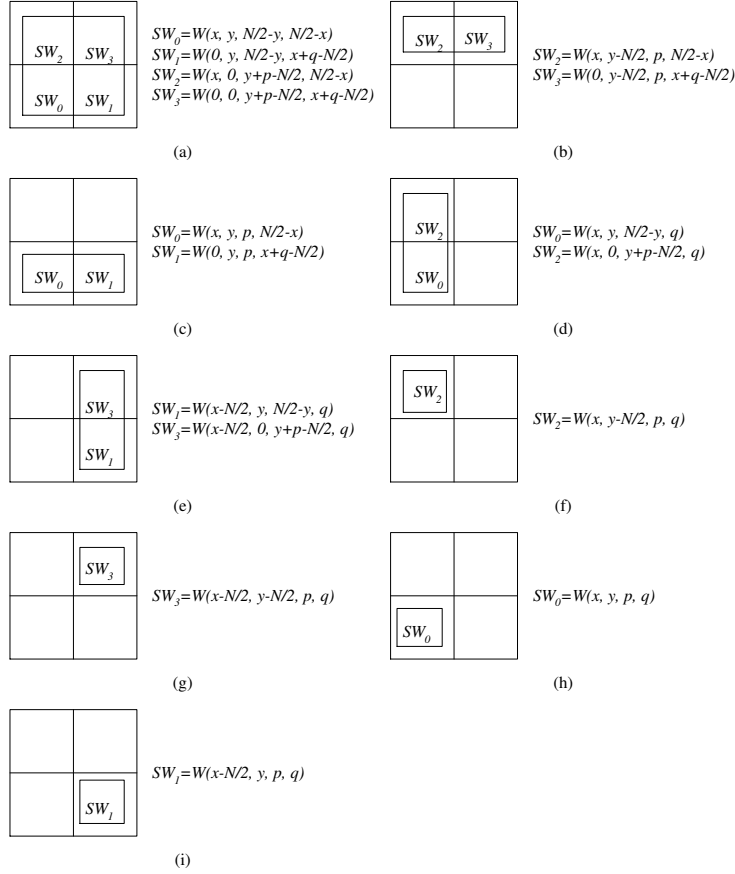


Figure 12: The sub-windows after partitioning the window for nine types: (a) Type 0; (b) Type 1; (c) Type 2; (d) Type 3; (e) Type 4; (f) Type 5; (g) Type 6; (h) Type 7; (i) Type 8.





































	Orientation A	Orientation B	Orientation C	Orientation D
Type 0				
Type 1				
Type 2				
Type 3				
Type 4				
Type 5				
Type 6				
Type 7				
Type 8				

Figure 13: 36 Cases

```

Procedure QSplitA (T, mino, x, y, p, q);
begin
  N2 := T/2;
  N4 := N2 * N2;
  type := determineType();

  case type of
    0 : QSplit(B, N2, mino, x, y, N2 - y, N2 - x);
        QSplit(A, N2, mino + N4, x, 0, y + p - N2, N2 - x);
        QSplit(A, N2, mino + N4 * 2, 0, 0, y + p - N2, x + q - N2);
        QSplit(D, N2, mino + N4 * 3, 0, y, N2 - y, x + q - N2);

    1 : QSplit(A, N2, mino + N4, x, y - N2, p, N2 - x);
        QSplit(A, N2, mino + N4 * 2, 0, y - N2, p, x + q - N2);

    2 : QSplit(B, N2, mino, x, y, p, N2 - x);
        QSplit(D, N2, mino + N4 * 3, 0, y, p, x + q - N2);

    3 : QSplit(B, N2, mino, x, y, N2 - y, q);
        QSplit(A, N2, mino + N4, x, 0, y + p - N2, q);

    4 : QSplit(A, N2, mino + N4 * 2, x - N2, 0, y + p - N2, q);
        QSplit(D, N2, mino + N4 * 3, x - N2, y, N2 - y, q);

    5 : QSplit(A, N2, mino + N4, x, y - N2, p, q);

    6 : QSplit(A, N2, mino + N4 * 2, x - N2, y - N2, p, q);

    7 : QSplit(B, N2, mino, x, y, p, q);

    8 : QSplit(D, N2, mino * 3, x - N2, y, p, q);
  end;
end;

```

Figure 14: Procedure *QSplitA*

```

Procedure QSplitB (T, mino, x, y, p, q);
begin
  N2 := T/2;
  N4 := N2 * N2;
  type := determineType();

  case type of
    0 : QSplit(A, N2, mino, x, y, N2 - y, N2 - x);
        QSplit(B, N2, mino + N4, 0, y, N2 - y, x + q - N2);
        QSplit(B, N2, mino + N4 * 2, 0, 0, y + p - N2, x + q - N2);
        QSplit(C, N2, mino + N4 * 3, x, 0, y + p - N2, N2 - x);

    1 : QSplit(B, N2, mino + N4 * 2, 0, y - N2, p, x + q - N2);
        QSplit(C, N2, mino + N4 * 3, x, y - N2, p, N2 - x);

    2 : QSplit(A, N2, mino, x, y, p, N2 - x);
        QSplit(B, N2, mino + N4 * 1, 0, y, p, x + q - N2);

    3 : QSplit(A, N2, mino, x, y, N2 - y, q);
        QSplit(C, N2, mino + N4 * 3, x, 0, y + p - N2, q);

    4 : QSplit(B, N2, mino + N4, x - N2, y, N2 - y, q);
        QSplit(B, N2, mino + N4 * 2, x - N2, 0, y + p - N2, q);

    5 : QSplit(C, N2, mino + N4 * 3, x, y - N2, p, q);

    6 : QSplit(B, N2, mino + N4 * 2, x - N2, y - N2, p, q);

    7 : QSplit(A, N2, mino, x, y, p, q);

    8 : QSplit(B, N2, mino + N4, x - N2, y, p, q);
  end;
end;

```

Figure 15: Procedure *QSplitB*

```

Procedure QSplitC (T, mino, x, y, p, q);
begin
  N2 := T/2;
  N4 := N2 * N2;
  type := determineType();

  case type of
    0 : QSplit(D, N2, mino, 0, 0, y + p - N2, x + q - N2);
        QSplit(C, N2, mino + N4, 0, y, N2 - y, x + q - N2);
        QSplit(C, N2, mino + N4 * 2, x, y, N2 - y, N2 - x);
        QSplit(B, N2, mino + N4 * 3, x, 0, y + p - N2, N2 - x);

    1 : QSplit(D, N2, mino, 0, y - N2, p, x + q - N2);
        QSplit(B, N2, mino + N4 * 3, x, y - N2, p, N2 - x);

    2 : QSplit(C, N2, mino + N4, 0, y, p, x + q - N2);
        QSplit(C, N2, mino + N4 * 2, x, y, p, N2 - x);

    3 : QSplit(C, N2, mino + N4 * 2, x, y, N2 - y, q);
        QSplit(B, N2, mino + N4 * 3, x, 0, y + p - N2, q);

    4 : QSplit(D, N2, mino, x - N2, 0, y + p - N2, q);
        QSplit(C, N2, mino + N4, x - N2, y, N2 - y, q);

    5 : QSplit(B, N2, mino + N4 * 3, x, y - N2, p, q);

    6 : QSplit(D, N2, mino, x - N2, y - N2, p, q);

    7 : QSplit(C, N2, mino + N4 * 2, x, y, p, q);

    8 : QSplit(C, N2, mino + N4, x - N2, y, p, q);
  end;
end;

```

Figure 16: Procedure *QSplitC*

```

Procedure QSplitD (T, mino, x, y, p, q);
begin
  N2 := T/2;
  N4 := N2 * N2;
  type := determineType();

  case type of
    0 : QSplit(C, N2, mino, 0, 0, y + p - N2, x + q - N2);
        QSplit(D, N2, mino + N4, x, 0, y + p - N2, N2 - x);
        QSplit(D, N2, mino + N4 * 2, x, y, N2 - y, N2 - x);
        QSplit(A, N2, mino + N4 * 3, 0, y, N2 - y, x + q - N2);

    1 : QSplit(C, N2, mino, 0, y - N2, p, x + q - N2);
        QSplit(D, N2, mino + N4, x, y - N2, p, N2 - x);

    2 : QSplit(D, N2, mino + N4 * 2, x, y, p, N2 - x);
        QSplit(A, N2, mino + N4 * 3, 0, y, p, x + q - N2);

    3 : QSplit(D, N2, mino + N4, x, 0, y + p - N2, q);
        QSplit(D, N2, mino + N4 * 2, x, y, N2 - y, q);

    4 : QSplit(C, N2, mino, x - N2, 0, y + p - N2, q);
        QSplit(A, N2, mino + N4 * 3, x - N2, y, N2 - y, q);

    5 : QSplit(D, N2, mino + N4, x, y - N2, p, q);

    6 : QSplit(C, N2, mino, x - N2, y - N2, p, q);

    7 : QSplit(D, N2, mino + N4 * 2, x, y, p, q);

    8 : QSplit(A, N2, mino + N4 * 3, x - N2, y, p, q);
  end;
end;

```

Figure 17: Procedure *QSplitD*

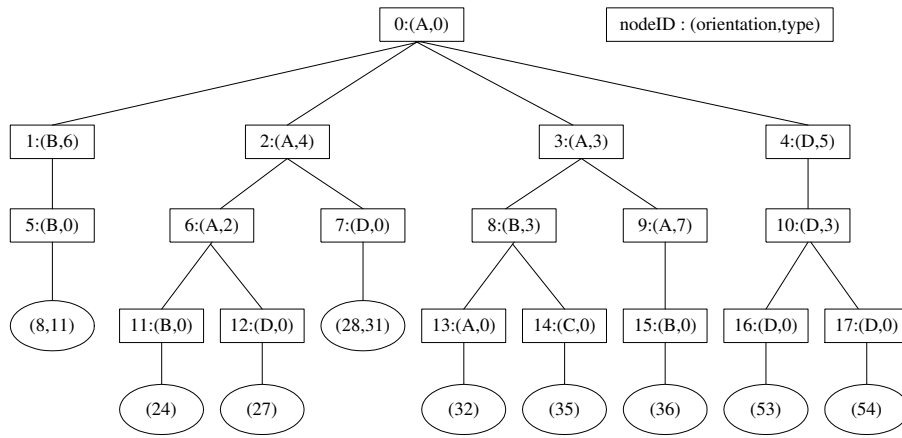


Figure 18: Trace of Sample 1

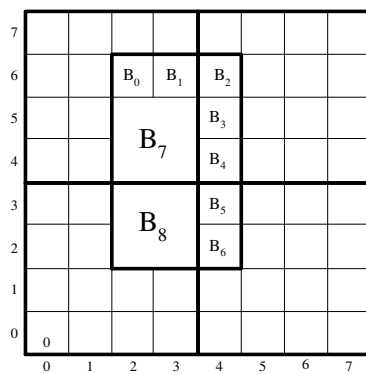


Figure 19: The maximal blocks of the query window for Sample 1

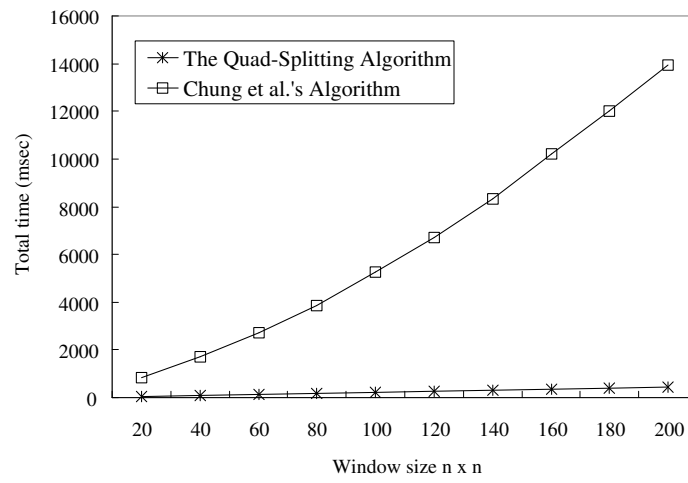


Figure 20: A comparison of the total execution time between the Quad-Aplitting algorithm and Chung *et al.*'s algorithm by varying the side length of the square window

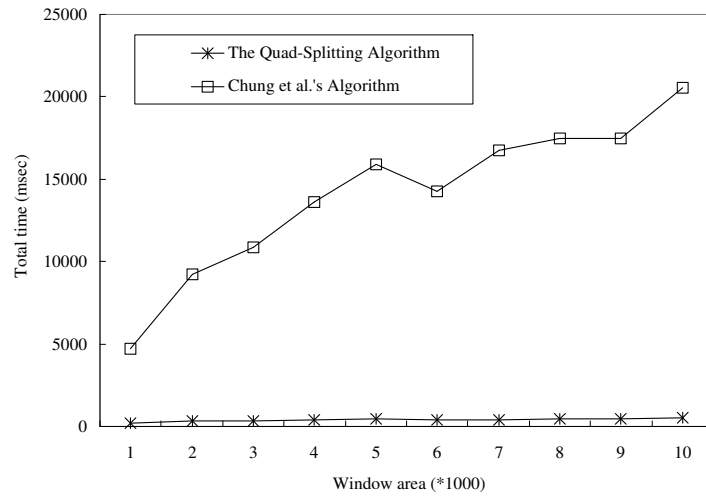


Figure 21: A comparison of the total execution time between the Quad-Splitting algorithm and Chung *et al.*'s algorithm by varying the size of the arbitrary rectangular window

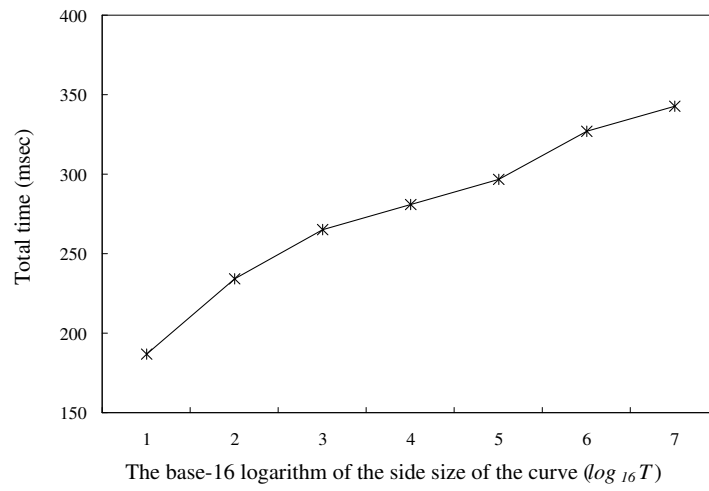


Figure 22: The total execution time of the Quad-Splitting algorithm for 100000 10×10 window queries by varying the side size of the Hilbert curve

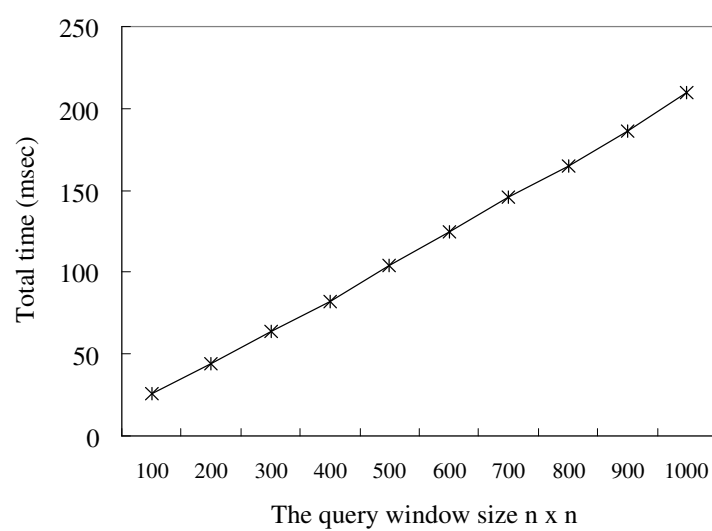


Figure 23: The total execution time of the Quad-Splitting algorithm for 1000 window queries on a 1024×1024 Hilbert curve by varying the size of the square window

List of Tables

1	Detail of nodes in Figure 17	46
2	Experimental results for square windows	47
3	Experimental results for arbitrary rectangular windows	48
4	Experimental results for $16^m \times 16^m$ Hilbert curves	49

Table 1: Detail of nodes in Figure 17

Node ID	$oABCD$	T	$mino$	x	y	p	q
0	A	8	0	2	2	5	3
1	B	4	0	2	2	2	2
2	A	4	16	2	0	3	2
3	A	4	32	0	0	3	1
4	D	4	48	0	2	2	1
5	B	2	8	0	0	2	2
6	A	2	24	0	0	1	2
7	D	2	28	0	0	2	2
8	B	2	32	0	0	2	1
9	A	2	36	0	0	1	1
10	D	2	52	0	0	2	1
11	B	1	24	0	0	1	1
12	D	1	27	0	0	1	1
13	A	1	32	0	0	1	1
14	C	1	35	0	0	1	1
15	B	1	36	0	0	1	1
16	D	1	53	0	0	1	1
17	D	1	54	0	0	1	1

Table 2: Experimental results for square windows

Window size n	T_{ours} (msec)	T_{Chung} (msec)	R
20	62	812	92.36
40	94	1716	94.52
60	124	2730	95.46
80	187	3869	95.17
100	218	5273	95.87
120	250	6708	96.27
140	296	8315	96.44
160	344	10218	96.63
180	390	12012	96.75
200	421	13931	96.98

Table 3: Experimental results for arbitrary rectangular windows

Window area n	T_{ours} (msec)	T_{Chung} (msec)	R
1000	219	4711	95.35
2000	296	9220	96.79
3000	343	10889	96.85
4000	374	13635	97.26
5000	452	15896	97.16
6000	390	14259	97.26
7000	422	16738	97.48
8000	468	17456	97.32
9000	437	17487	97.50
10000	514	20545	97.50

Table 4: Experimental results for $16^m \times 16^m$ Hilbert curves

T	$\log_{16} T$	Execution time (msec)
16	1	187
16^2	2	234
16^3	3	265
16^4	4	281
16^5	5	297
16^6	6	327
16^7	7	343