Mathematical Biosciences and Engineering

MBE, 20(12): 20668–20682.
DOI: 10.3934/mbe.2023914
Received: 08 September 2023
Revised: 02 November 2023
Accepted: 06 November 2023
Published: 16 November 2023

http://www.aimspress.com/journal/MBE

Research article

Efficient entry point encoding and decoding algorithms on 2D Hilbert space filling curve

Mengjuan Li[1], Yao Fan[2], Shaowen Sun[2], Lianyin Jia[2,*] and Teng Liang[3,*]

[1] Library, Yunnan Normal University, Kunming 650500, China
[2] Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, China
[3] School of Communications Information Engineering, Yunnan Communications Vocational and Technical College, Kunming 650500, China

*** Correspondence:** Email: lianyinjia@kust.edu.cn, 13888990971@139.com.

Abstract: The Hilbert curve is an important method for mapping high-dimensional spatial information into one-dimensional spatial information while preserving the locality in the high-dimensional space. Entry points of a Hilbert curve can be used for image compression, dimensionality reduction, corrupted image detection and many other applications. As far as we know, there is no specific algorithms developed for entry points. To address this issue, in this paper we present an efficient entry point encoding algorithm (EP-HE) and a corresponding decoding algorithm (EP-HD). These two algorithms are efficient by exploiting the m consecutive 0s in the rear part of an entry point. We further found that the outputs of these two algorithms are a certain multiple of a certain bit of s, where s is the starting state of these m levels. Therefore, the results of these m levels can be directly calculated without iteratively encoding and decoding. The experimental results show that these two algorithms outperform their counterparts in terms of processing entry points.

Keywords: Hilbert space filling curve; entry point; EP-HE; EP-HD

1. Introduction

The Hilbert space filling curve (HSFC) is an important method for mapping high-dimensional spatial information into one-dimensional spatial information while preserving the locality in the

high-dimensional space. As it has the properties of simple, space filling, self-similar and self-avoiding, it has been widely used in various fields such as deep learning [1–5], spatial query processing [6,7], image processing [3,4,8], location privacy protection [9] and cache locality preserving [10].

Taking deep learning as an example, Hilbert curves can be used to transform 1D data to 2D, 2D data to 1D or to sort 2D data. Hilbert curves are used to transform mammographic images into 1D vectors to further detect breast cancer [5]. In paper [1], the authors converted volumetric data into 1D vectors, then used convolutional neural networks (CNN) to process these vectors. Differently, 1D surface electromyography (sMEG) and DNA data were also transformed into 2D images by Hilbert curves for further processing [2,3]. Bappy et al. [4] ordered the image patches using Hilbert curves to better preserve their spatial locality and then fed the patches into a Long Short-Term Memory (LSTM) to detect the image forgeries.

The Hilbert curve and Z curve [11] are two typical space filling curves. In this paper, we focus on the Hilbert curve, as it has much higher spatial locality [12]. Considering the extensive application of Hilbert curves, instead of researching the specific application fields, we focus on 2D HSFC encoding and decoding, which is the fundamental basis of these applications. The performances of HSFC encoding and decoding has a direct impact on these application. Currently, the encoding and decoding efficiencies are not high due to the complex mapping rules of HSFC. It is important to improve the encoding and decoding efficiency to boost the efficiencies of HSFC based applications.

The HSFC encoding and decoding algorithms can be divided into recursive based algorithms and iterative based algorithms. Among the recursive based algorithms, the byte-oriented algorithm proposed by Butz et al. [13] is a classic one. However, recursive based algorithms require large mapping overheads and are generally less efficient. Therefore, recent researches are mostly focusing on iterative based.

Among the iterative based algorithms, Moore et al. [14] transformed the recursive algorithm in [13] into a iterative one. The algorithm proposed by Burkardt et al. [15] was recommended by Wikipedia, which introduced additional computing to adjust the input data iteratively. These two algorithms are not efficient as they require high input data adjustment overheads. Bohm et al. [10] proposed a non-recursive Lindenmayer algorithm, which is also transformed from a recursive algorithm. However, this algorithm can only work to decode data along the Hilbert curve in a window.

In recent years, the state-view based algorithms [16,17] have been extensively studied for their simplicity, intuitiveness and efficiency in nature. Li et al. [16] proposed a state transfer matrix based iterative encoding algorithm with complexity $O(k)$, where k is the total number of levels. Encoding and decoding algorithms on skew-distributed data researched in [18]. Existing algorithms need separate state views for encoding and decoding. To decrease space overheads, a universal state based algorithm was proposed in [17], which used the same state-view for both encoding and decoding by utilizing quadrant mapping before encoding and inverse mapping after decoding. Encoding and decoding algorithms on three-dimensional or high-dimensional Hilbert curves were researched in [19–21].

A Hilbert curve divides the entire Hilbert space into hierarchically organized subspaces, as we will discuss in Section 2. The first point of a subspace where the curve enters into is called an entry point (EP) of that subspace. An EP can be viewed as a representing point of the subspace it belongs to; thus, the whole subspace can be compressed into a single point. Consequently, the entire Hilbert space can be compressed into a series of EPs, substantially decreasing the scale of original data. In this sense, EPs can be used to compress images, reduce the dimensionality of images, or potentially help researchers detect corrupted or forged images more efficiently.

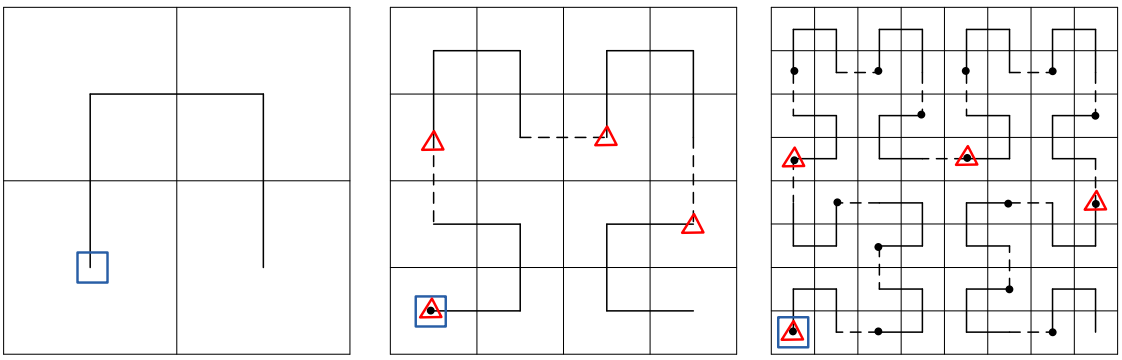
Even though EPs are useful in many real scenarios, surprisingly, as far as we know, none of the aforementioned works have researched them before, let alone the encoding and decoding of EPs. To address this issue, we are devoted to improving the encoding and decoding efficiencies of EPs in Hilbert space. One important characteristic of EP is that the rear m levels of an EP are consecutive 0s. Given this characteristic, it is inefficient to follow traditional level-wise encoding or decoding algorithms, e.g., Li's [16]. More efficient algorithms need to be specifically developed for EPs.

By exploiting this characteristic of EPs, in this paper we found the computed outputs (either Hilbert codes for encoding or coordinates for decoding) are a certain multiple of a certain bit of s, where s is the starting state of these m levels. Therefore, the output of these m levels can be directly calculated without iteratively encoding and decoding. Based on these findings, an EP Hilbert encoding algorithm (EP-HE) and the corresponding EP Hilbert decoding (EP-HD) algorithm are proposed by exploiting efficient bit manipulations. Experiments show that the EP-HE and EP-HD algorithms outperform existing algorithms.

2. HSFC

HSFC, first proposed by the German mathematician Hilbert, is a continuous non-differentiable fractal space-filling curve that can linearly run through each discrete cell of a high-dimensional (two-dimensional and above) space, thus effectively mapping the high-dimensional spatial information into one-dimensional and better preserving the localities of the high-dimensional space.

An HSFC passes through the center of each subspace once and only once without interruption and self-intersection. The sequence number for a subspace that the HSFC passes through is the Hilbert code of that subspace. A HSFC of level 1 (HC1) divides the entire space into four small subspaces, namely, the lower left, upper left, lower right and upper right. These four subspaces correspond to quadrant I, quadrant II, quadrant III, and quadrant IV, respectively. By placing a HC1 in each quadrant and flipping it 90° to the left for quadrant I and 90° to the right for quadrant III, a HC2 can be obtained. Following this way, a HCk can be easily derived. The HC1, HC2 and HC3 are shown in Figure 1(a)–(c), respectively.

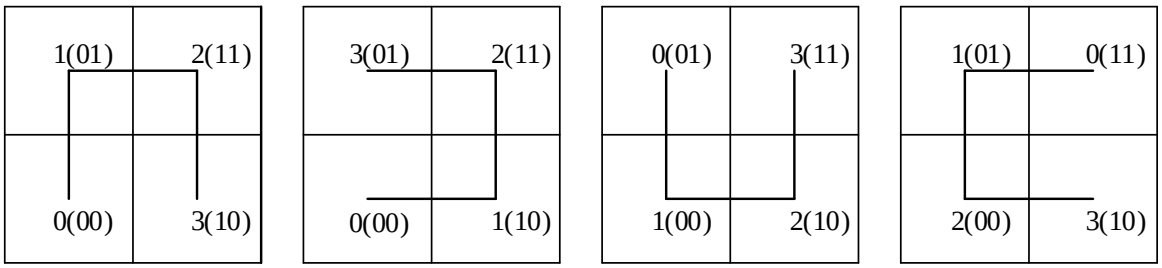


(a) HC1 (b) HC2 (c) HC3

Figure 1. The Hilbert curves for HC1, HC2 and HC3.

For HC1, there are four different opening directions as shown in Figure 2, corresponding to the four basic states 0 (opening downward), 1 (opening leftward), 2 (opening upward), and 3 (opening

rightward), respectively. The self-similarity and self-replication properties of HSFC determine that any level of HSFC can be derived iteratively or recursively from these four basic states. For ease of description, we assume the state of the first level of a HSCF is always 0, which means it is opening downward.



(a) state 0 (b) state 1 (c) state 2 (d) state 3

Figure 2. The four basic states.

From the discussion above, a Hilbert curve divides the entire Hilbert space into hierarchically organized subspaces, with the *i*-th ($1 \le i \le k$) level containing 4^i subspaces. For a space Ω, an EP is essentially the first subspace of Ω that a Hilbert curve enters into. There are 4^{i-1} EPs in the *i*-th level, so there are $4^0 + 4^1 + ... + 4^{k-1} = (4^k - 1)/3$ EPs for all the levels in total. In this paper, we call an EP in the *i*-th level an *i*-EP. For example, there are one *1*-EP (in blue square), four *2*-EPs (in red triangle) and 16 3-EPs (in black dot) in Figure 1(c). Note that an *i*-EP is also an (*i*+1)-EP, (*i*+2)-EP,…or (*k*-1)-EP, so we use the smallest *i* for an EP when there is no ambiguity.

One important characteristic of EP lies in that the rear $m = k-i+1$ levels of an *i*-EP are consecutive zeros for a HC*k*. For example, the coordinate of the third 2-EP *P* in HC3 is (100_2, 100_2) with zeros in the rear two levels. The Hilbert code of this point is 100000_2 (the subscript two means binary form), which also has two consecutive level of zeros in the rear part.

Given this characteristic, it is inefficient to follow traditional level-wise encoding or decoding algorithms. More efficient algorithms need to be specifically developed for encoding or decoding EPs.

For an EP *P* with the rear *m* levels consecutive zeros, the coordinate of *P* is denoted as $C_P = (X,Y)$, where $X = (x_1x_2\ldots x_{k-m}x_{k-m+1}\ldots x_k)_2$, $Y = (y_1y_2\ldots y_{k-m}y_{k-m+1}y_{k-m+2}\ldots y_k)_2$ are its coordinate components with $X^m = (x_{k-m+1}\ldots x_k)_2 = 0$ and $Y^m = (y_{k-m+1}y_{k-m+2}\ldots y_k)_2 = 0$. Similarly, the Hilbert code of *P* is denoted as $H_P = (h_1\ldots h_{2k-2m}\, h_{2k-2m+1}\ldots h_{2k})_2$ with $H^m = (h_{2k-2m+1}\ldots h_{2k})_2 = 0$. Here, x_i corresponds to the *i*-th level of *X* and $h_{2i-1}h_{2i}$ corresponds to the *i*-th level of the Hilbert code of *H* (H_i for short).

3. HSFC encoding and decoding

3.1. The encoding state-views

In the encoding stage, according to Figure 2 and the rotation and flipping rules of Hilbert curve,

we design two mapping state-views, namely, coordinate-Hilbert mapping (CHM) and coordinate-state mapping (CSM), as shown in Tables 1 and 2, respectively. CHM is designed as a three-dimensional array and is responsible for mapping the coordinates of the i-th level (x_iy_i) to H_i with the help of the i-th state s_i, e.g., H_i = CHM[s_i][x_i][y_i] . CSM is also a three-dimensional array and is responsible for mapping the coordinates of the i-th level (x_iy_i) to the corresponding state of the (i+1)-th level s_{i+1}, e.g., s_{i+1} = CSM[s_i][x_i][y_i].

Table 1. CHM.

State\Coordinate	00	01	10	11
0	00	01	11	10
1	00	11	01	10
2	10	11	01	00
3	10	01	11	00

Table 2. CSM

State\Coordinate	00	01	10	11
0	00	01	11	10
1	00	11	01	10
2	10	11	01	00
3	10	01	11	00

3.2. EP-HE

As mentioned before, this paper focuses on EPs of which the rear m levels are consecutive 0s. The core of EP-HE algorithm is based on Theorem 1, which is described as follows.

Theorem 1: Given $X^m = x_{k-m+1}x_{k-m+2}\ldots x_k$ =0, $Y^m = y_{k-m+1}y_{k-m+2}\ldots y_k$ = 0 and $A = 2*(2^{2m}-1)/3$, the H^ms for s_{k-m+1} = 0, 1, 2 and 3 are 0, 0, A and A, respectively.

Proof: We prove it in the following four cases.

For s_{k-m+1} = 0, as $x_{k-m+1}y_{k-m+1} = 00_2$, we have $H_{k-m+1} = 00_2$ and s_{k-m+2} = 1 by retrieving CHM and CSM. As $x_{k-m}y_{k-m} = 00_2$, we have $H_{k-m+2} = 00_2$ and s_{k-m+3} = 0. The s_{k-m+i} cycles between 0 and 1 and H_{k-m+i} are always 00_2 for $1 \le i \le m$. As a result, H^m =0 holds.

For s_{k-m+1} = 1, the s_{k-m+i} cycles between 1 and 0 and $H_{k-m+i} = 00_2$ always holds for $1 \le i \le m$. As a result, H^m = 0 holds.

For s_{k-m+1} = 2, the s_{k-m+i} cycles between 2 and 3 and $H_{k-m+i} = 10_2$ always holds for $1 \le i \le m$ in a similar way. As a result, $H^m = A = (10\ldots1010)_2 = 2*(2^{2m}-1)/3$ holds.

For s_{k-m+1} = 3, the s_{k-m+i} cycles between 3 and 2 and $H_{k-m+i} = 10_2$ always holds for $1 \le i \le m$ in a similar way. As a result, $H^m = A = (10\ldots1010)_2 = 2*(2^{2m}-1)/3$ holds.

Thus, Theorem 1 holds.

From Theorem 1, when X^m and Y^m are all 0s, H^m only has two possible values and can be computed immediately. Therefore, level-wise iterative encoding for these m levels can be avoided, thus improving the efficiencies.

To boost the efficiencies of EP-HE, we introduce the following two optimizations, each of which works on high efficient bit manipulations. This makes our algorithms execute really fast in

real scenarios.

1) We notice that there will introduce excessive selection costs if we use if-else statements to determine the value of rear m levels. Note that H^m depends on the value of s_{k-m+1}, specifically the left bit of the two bits of s_{k-m+1}. So, $(s_{k-m+1}>>1)*A$ can be used to determine H^m, so excessive if-else statements can be avoided. Here, ">>" means right logical shift.

2) To quickly detect the number of 0s in the rear part of X and Y, i.e., m, we introduce the efficient last bit detection (LBD) algorithm, which works on $X \& (-X)$ to get the position of the last set bit in X. To avoid executing LBD twice for both X and Y, we execute it for $X|Y$ instead; thus, the efficiency can be further improved.

Based on the aforementioned description, the implemented EP-HE algorithm is presented in Algorithm 1.

Algorithm 1. EP-HE.

Input: X: Horizontal coordinate component
Y: Vertical coordinate component
k: the number of levels
Output: H: Hilbert code

1. $H = 0$, $s = 0$
2. m = LBD($X|Y$)
3. for $i = 1$ to k-m
4. $H = H << 2 \mid \text{CHM}[s][x_i][y_i]$
5. $s = \text{CSM}[s][x_i][y_i]$
6. $A = ((1<<2m)-1)/3*2$
7. $H = (H<<2m) \mid (s>>1)*A$

3.3. The decoding state-views

In the decoding stage, two decoding mapping state-views, Hilbert-coordinate mapping (HCM) and Hilbert-state mapping (HSM), are created as shown in Tables 3 and 4, respectively. HCM is designed as a two-dimensional array and is responsible for mapping H_i to x_iy_i with the help of s_i, e.g., $x_iy_i = \text{HCM}[s_i][H_i]$. HSM is also a two-dimensional array and is responsible for mapping H_i to s_{i+1}, e.g., $s_{i+1} = \text{HSM}[s_i][H_i]$.

Table 3. HCM.

State\Coordinate	00	01	10	11
0	00	01	11	10
1	00	10	11	01
2	11	10	00	01
3	11	01	00	10

Table 4. HSM.

State\Coordinate	00	01	10	11
0	1	0	0	3
1	0	1	1	2
2	3	2	2	1
3	2	3	3	0

3.4. EP-HD

Similar to encoding, the core of EP-HD algorithm is based on Theorem 2, which is described as follows.

Theorem 2: If $H^m = (h_{2k-2m+1}\ldots h_{2k})_2 = 0$ and $B = 2^m$-1, then X^m = 0, 0, B, B and Y^m = 0, 0, B, B for s_{k-m+1} = 0, 1, 2 and 3, respectively.

Proof: We prove it in the following four cases.

For $s_{k-m+1} = 0$, as $h_{2k-2m+1}h_{2k-2m+2} = 00_2$, we have $x_{k-m+1} = 0$, $y_{k-m+1} = 0$ and $s_{k-m+2} = 1$ by retrieving HCM and HSM. As $h_{2k-2m+3}h_{2k-2m+4} = 00_2$, we have $x_{k-m+2} = 0$, $x_{k-m+2} = 0$ and $s_{k-m+3} = 0$. The s_{k-m+i} cycles between 0 and 1, and $X_{k-m+i} = 0$ and $Y_{k-m+i} = 0$ always hold. As a result, $X^m = 0$ and $Y^m = 0$ hold.

For $s_{k-m+1} = 1$, the s_{k-m+i} cycles between 1 and 0, and $X_{k-m+i} = 0$ and $Y_{k-m+i} = 0$ always hold for $1 \le i \le m$. As a result, $X^m = Y^m = 0$.

For $s_{k-m+1} = 2$, the s_{k-m+i} cycles between 2 and 3, and $X_{k-m+i} = 1$ and $Y_{k-m+i} = 1$ always hold for $1 \le i \le m$. As a result, $X^m = Y^m = B = (1\ldots11)_2 = 2^m$-1.

For $s_{k-m+1} = 3$, the s_{k-m+i} cycles between 3 and 2, and $X_{k-m+i} = 1$and $Y_{k-m+i} = 1$ always hold for $1 \le i \le m$. As a result, $X^m = Y^m = B = (1\ldots11)_2 = 2^m$-1.

Thus, Theorem 2 holds.

From Theorem 2 we can compute X^m and Y^m directly, and iterative decoding for these m levels can be avoided. As in encoding, EP-HD uses $(s_{k-m+1}>>1)*B$ to compute X^m and Y^m to avoid the costs of excessive if-else statements. The LBD algorithm can also be used to get the position of the last set bit in H. The implemented EP-HD algorithm is shown in Algorithm 2.

Algorithm 2. EP-HD.

Input: H: Hilbert code
k: the number of levels
Output: X: Horizontal coordinate component
Y: Vertical coordinate component

1. $X = 0$, $Y = 0$, $s = 0$
2. m = LBD(H)
3. for $i = 1$ to k-m
4. $X = (X<<1) \mid \text{HCM}[s][H_i]>>1$
5. $Y = (Y<<1) \mid \text{HCM}[s][H_i]$ &1
6. $s = \text{HSM}[s][H_i]$
7. $B = (1<<m)-1$
8. $X = (X<<\text{m}) \mid B*(s>>1)$
9. $Y = (Y<<\text{m}) \mid B*(s>>1)$

3.5. Complexity analyses

Compared with Li's [16], which needs to iteratively encode all the levels and has a complexity of O(k), EP-HE and EP-HD only need to encode or decode O(*k*-*m*) levels; thus, their time complexity O(*k*-*m*). In terms of space complexity, we need the storage to store the four state-views: CHM, CSM, HCM and SCM. In this paper, we use char to store the values in these state-views, so each state-view needs 16 bytes, leading to 64 bytes in total for all the state-views.

4. Experimental results

4.1. Environment

Hardware platform: CPU: Intel® Core ™ i5-6300HQ@2.30GHz, RAM: 8GB. Software environment: windows 11 64-bit, Microsoft Visual Studio C++ 2019.

4.2. Datasets

In this paper, we use both synthetic datasets and real datasets.

For the synthetic datasets, considering EPs are essentially skewed data skewed with consecutive 0s in the rear *m* levels, we generate skewed datasets by setting three parameters *m*, *β*, *k*. Here, *k* denotes the number of total levels, *m* denotes the input data with the rear *m* levels of all 0s, and β denotes the percentage of data with the rear *m* levels of all 0s to the total data in the data set. For example, $k = 16$, $m = 4$ and $\beta = 50\%$ represents each input data at 16 levels and 50% of the input data with the rear four levels all 0. For each parameter combination, we generate 10 million skewed coordinates as our default dataset.

For real dataset, we use T-Drive trajectory [22] dataset collected by Microsoft. This dataset contains one-week trajectories of 10,357 taxis in Beijing. We choose a zone by setting (116.397221 E, 39.90960 N) as the center and enlarging 0.2° on four sides, thus selecting 13,987,446 coordinates. For each coordinate component with float value, we process it into arbitrary *n* level integers by iteratively detecting its falls into what half part. For example, if a coordinate component X falls into the right half part of the zone, we set the first level 1, then execute the same operations on the right half part untill *n* is reached.

Note that we directly use the datasets above for encoding algorithms and then use the encoded data for decoding algorithms.

To illustrate the effectiveness of our algorithms, we compare EP with the following five algorithms: first zeros free (FZF) [18], Burkardt [15], Moore [14], uniState [17] and Li [16]. For each algorithm, we add a suffix "-HE" and a suffix "-HD" for the encoding and decoding algorithms, respectively.

4.3. Experiments on synthetic datasets

4.3.1 Encoding

We use the following settings to evaluate all the encoding algorithms.

1) We use the datasets generated with setting $k = 32$, $\beta = 50\%$ and vary *m* from 4 to 24. The

effects of m on encoding are shown in Figure 3.

As seen in Figure 3, EP-HE outperforms the other algorithms when k = 32, β = 50% and the encoding time of EP-HE gradually decreases with the increase of m, whereas m has little impact on Li-HE. This is because as m increases, the number of levels need to be iteratively encoded for the EP-HE decreases. When m = 24, EP-HE runs 10.38% faster than Li's and 3.10x faster than Burkardt's. In total, except UniState-HE, which needs more time for quadrant mapping and inverse mapping, state-view based algorithms outperform non-state-view based ones. The results show the advantages of state-view based algorithms.

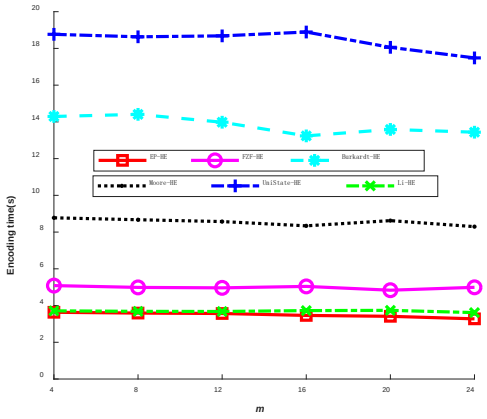


Figure 3. The effects of m on encoding.

2) We use the datasets generated with setting k = 32, m = 12 and vary β from 20 to 100%. The effects of β on encoding are shown in Figure 4.

As can be seen from Figure 4, EP-HE outperforms the other algorithms. When k = 32, m = 12, the encoding time for EP-HE gradually decreases with the increase of β. This shows that the more input data with rear m levels equals 0, the higher the efficiency will be. The encoding times for EP-HE and Li's are 3.206 and 3.729 s, respectively when β = 100%. In this case, EP-HE runs 16.31% faster than Li's and 3.26x faster than Burkardt's.

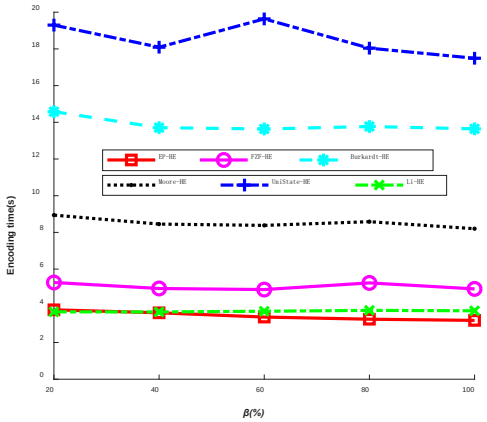


Figure 4. The effects of β on encoding.

3) We use the datasets generated with setting m = 12, β = 50% and vary k from 16 to 32. The effects of k on encoding are shown in Figure 5.

As can be seen from Figure 5, the levels needed to be iterative encoded for all the algorithms increase with the increase of k; thus, the overall encoding times for all algorithms tend to increase. EP-HE outperforms the other algorithms and the encoding time of EP-HE grows at a relatively lower rate than the other algorithms. For example, the encoding times for EP-HE are 1.829, 2.601 and 3.488 s for k = 16, 24 and 32, respectively, whereas the corresponding encoding times of Li's are 2.203, 2.694 and 3.814 s, respectively.

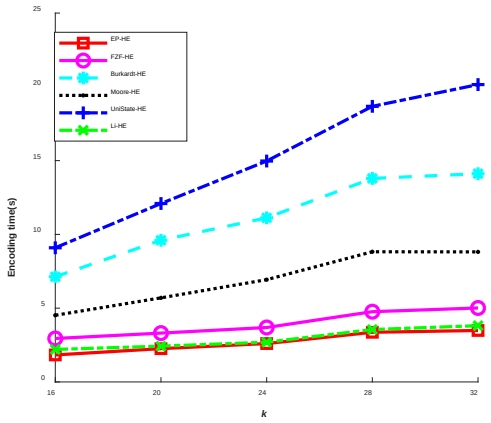


Figure 5. The effects of k on encoding.

4.3.2 Decoding

Similarly, we use the following settings to evaluate all the encoding algorithms.

1) We use the datasets generated with setting k = 32, β = 50% and vary m from 4 to 24. The effects of m on decoding are shown in Figure 6.

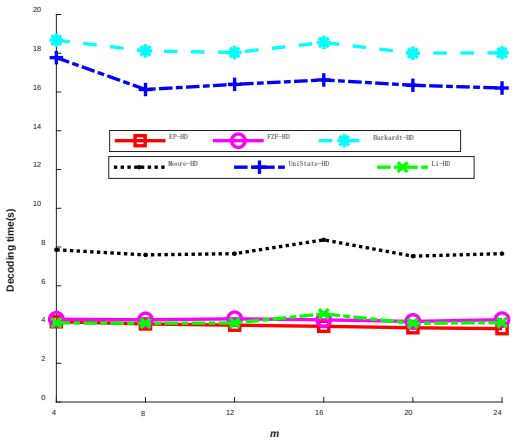


Figure 6. The effects of m on decoding.

As seen in Figure 6, the number of levels needed to be iteratively decoded for EP-HD decrease with the increase of m; thus, the decoding time of EP-HD gradually decreases as well. In this scenario, EP-HD performs the best among all the decoding algorithms. When m = 24, EP-HD only needs 3.786 s to decode all 10 million Hilbert codes and runs 8.24% faster than Li's and 3.76x faster than Burkardt's.

2) We use the datasets generated with setting k = 32, m = 12 and vary β from 20 to 100%. The effects of β on decoding are shown in Figure 7.

As seen in Figure 7, EP-HD outperforms the other algorithms and the decoding time of EP-HD gradually decreases with the increase of β. This is because fewer levels need to be iteratively decoded as β increases. When β = 100%, EP-HD only needs 3.836 s decoding time and runs 16.12% faster than Li's and 3.73x faster than Burkardt's.

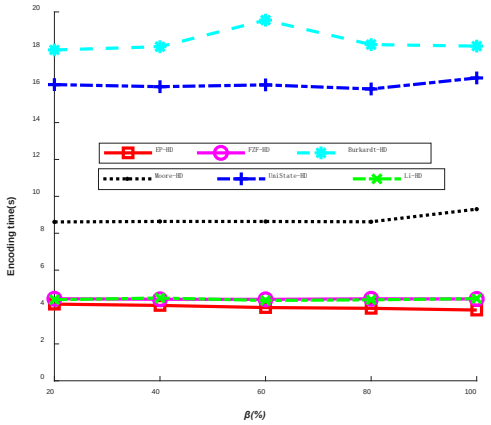


Figure 7. The effects of β on decoding.

3) We use the datasets generated with setting m = 12, β = 50% and vary k from 16 to 32. The effects of k on decoding are shown in Figure 8.

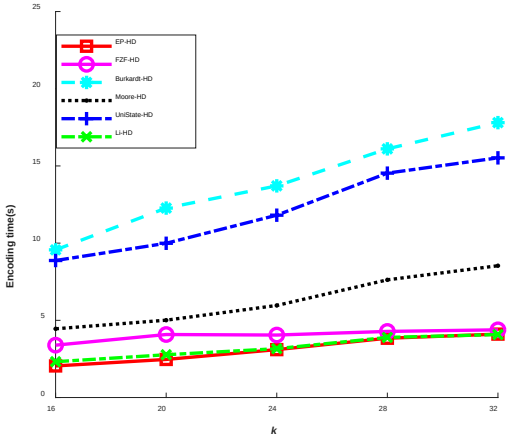


Figure 8. The effect of k on decoding.

As seen in Figure 8, the decoding time of all algorithms tends to increase with the increase of *k*. As *k* increases, the levels to be decoded increase as well, leading to a larger decoding time. Similarly, EP-HD runs faster than the others.

4.4. Experiments on real dataset

To illustrate the performance of our algorithms on real dataset, we carry out experiments on T-Drive. We process each data to n = 4, 8, 16, 24, 32 levels separately and then fill 0s in its rear until the total number of levels k reaches 32, leading to m = 0, 8, 16, 24, 32, respectively. The encoding and decoding results are shown in Figures 9 and 10, respectively.

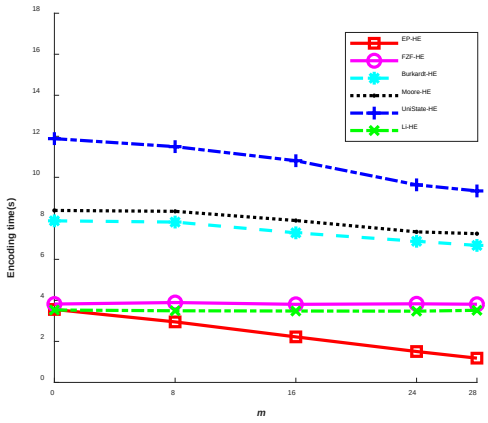


Figure 9. The effect of *m* on encoding.

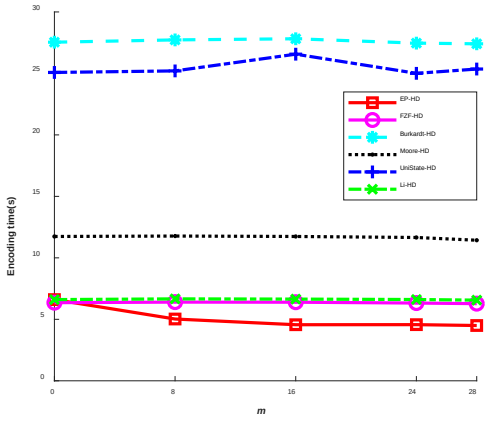


Figure 10. The effect of *m* on decoding.

From Figure 9, we can see when m = 0, EP-HE degenerates to Li-HE as we need to encode all the levels. With the increase of m, the encoding time of EP-HE decreases sharply, leading to a significant improvement compared to the other algorithms. When m = 28, EP-HP needs 1.183 s to encode all the coordinates, whereas Li-HE needs 3.516 s. EP-HP is 1.97x faster Li-HE. EP-HP performs better on T-Drive than synthetic datasets, which lies in that we encode the coordinates

along the trajectory; thus, the consecutive coordinates are very close and have a large number of common levels. From Figure 10, we can draw similar conclusions. However, the decoding improvements are not as high as the encoding ones, which lies in that we need to execute LBD on 64 bit Hilbert codes, compared with LBD on 32b coordinate components.

5. Conclusions

In this paper, by exploiting efficient state-views and bit manipulations, we presented an efficient entry point encoding algorithm EP-HE and decoding algorithm EP-HD, which can avoid encoding or decoding the rear *m* levels with consecutive 0s, thus, significantly improving the encoding and decoding efficiency of entry points. Experimental results show the superiorities of EP-HE and EP-HD. Next, we plan to apply our algorithms on deep learning based image processing tasks [23–25] and spatial or texture processing tasks [26,27].

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This research was supported by the National Natural Science Foundation of China [No 62262035, 62262034].

Conflict of interest

The authors declare there is no conflict of interest.

References

1. T. Corcoran, R. Zamora-Resendiz, X. Liu, S. Crivelli, A spatial mapping algorithm with applications in deep learning-based structure classification, preprint, arXiv: 1802.02532.
2. B. Yin, M. Balvert, D. Zambrano, A. Schönhuth, S. Bohte, An image representation based convolutional network for DNA classification, preprint, arXiv: 1806.04931.
3. P. Tsinganos, B. Cornelis, J. Cornelis, B. Jansen, A. Skodras, Hilbert sEMG data scanning for hand gesture recognition based on deep learning, *Neural Comput. Appl.*, **33** (2021), 2645–2666. https://doi.org/10.1007/s00521-020-05128-7
4. J. H. Bappy, C. Simons, L. Nataraj, B. S. Manjunath, A. K. Roy-Chowdhury, Hybrid LSTM and encoder–decoder architecture for detection of image forgeries, *IEEE Trans. Image Process.*, **28** (2019), 3286–3300. https://doi.org/10.1109/TIP.2019.2895466
5. S. Dhahbi, W. Barhoumi, J. Kurek, B. Swiderski, M. Kruk, E. Zagrouba, False-positive reduction in computer-aided mass detection using mammographic texture analysis and classification, *Comput. Methods Programs Biomed.*, **160** (2018), 75–83. https://doi.org/10.1016/j.cmpb.2018.03.026

6. Z. Yao, J. Zhang, T. Li, Y. Ding, A trajectory big data storage model incorporating partitioning and spatio-temporal multidimensional hierarchical organization, *ISPRS Int. J. Geo-Inform.*, **11** (2022), 621. https://doi.org/10.3390/ijgi11120621
7. Z. Liu, L. Wu, W. Meng, H. Wang, W. Wang, Accurate range query with privacy preservation for outsourced location-based service in IOT, *IEEE Int. Things J.*, **8** (2021), 14322–14337. https://doi.org/10.1109/JIOT.2021.3068566
8. X. Zhang, L. Wang, Z. Zhou, Y. Niu, A chaos-based image encryption technique utilizing Hilbert curves and H-fractals, *IEEE Access*, **7** (2019), 74734–74746. https://doi.org/10.1109/ACCESS.2019.2921309
9. P. Li, Z. Xie, Z. Zhou, G. Yue, S. Zheng, X. Yang, A source-location privacy preservation method based on hilbert-filling-curve routing protocol in marine wireless sensor networks (in Chinese), *J. Electron. Inform. Technol.*, **42** (2020), 1510–1518.
10. C. Böhm, M. Perdacher, C. Plant, A novel hilbert curve for cache-locality preserving loops, *IEEE Trans. Big Data*, 2018.
11. L. Jia, H. Tang, M. Li, B. Zhao, S. Wei, H. Zhou, An efficient association rule mining-based spatial keyword index, *Int. J. Data Warehous. Mining*, **19** (2023), 1–19. https://doi.org/10.4018/IJDWM.316161
12. J. Chen, L. Yu, W. Wang, Hilbert space filling curve based scan-order for point cloud attribute compression, *IEEE Trans. Image Process.*, **31** (2022), 4609–4621. https://doi.org/10.1109/TIP.2022.3186532
13. A. R. Butz, Alternative algorithm for Hilbert's space-filling curve, *IEEE Trans. Comput.*, **100** (1971), 424–426. https://doi.org/10.1109/T-C.1971.223258
14. Fast Hilbert Curve Generation, Sorting, and Range Queries. Available from: https://github.com/Cheedoong/hilbert.
15. Convert between 1D and 2D coordinates of Hilbert Curve. Available from: http://people.math.sc.edu/Burkardt/c_src/hilbert_curve/hilbert_curve.html.
16. S. Li, E. Zhong, S. Wang, An algorithm for Hilbert ordering code based on state-transition matrix (in Chinese), *J. Geo-Inform. Sci.*, **16** (2014), 846–851.
17. Hilbert_spatial_index. Available from: https://github.com/xcTorres/hilbert_spatial_index.
18. L. Jia, M. Chen, M. Li, J. You, J. Ding, State view based efficient Hilbert encoding and decoding algorithms (in Chinese), *J. Electron. Inform. Technol.*, **42** (2020), 1494–1501.
19. J. Zhang, S. Kamata, A generalized 3-D Hilbert scan using look-up tables, *J. Visual Commun. Image Represent.*, **23** (2012) , 418–425. https://doi.org/10.1016/j.jvcir.2011.12.005
20. L. Jia, B. Liang, M. Li, Y. Liu, Y. Chen, J. Ding, Efficient 3D Hilbert curve encoding and decoding algorithms, *Chinese J. Electron.*, **31** (2022), 1–8. https://doi.org/10.1049/cje.2020.00.171
21. H. Liu, T. Cui, W. Leng, L. Zhang, Encoding and decoding algorithms for arbitrary dimensional Hilbert order, preprint, arXiv: 1601.01274.
22. T-Drive trajectory data sample. Available from: https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/.
23. C. Tian, Y. Zhang, W. Zuo, C. W. Lin, D. Zhang, Y. Yuan, A heterogeneous group CNN for image super-resolution, *IEEE Trans. Neural Networks Learn. Syst.*, 2022. https://doi.org/10.1109/TNNLS.2022.3210433

24. Q. Zhang, J. Xiao, C. Tian, J. C. Lin, S. Zhang, A robust deformed convolutional neural network (CNN) for image denoising, *CAAI Trans. Intell. Technol.*, **8** (2023), 331–342. https://doi.org/10.1049/cit2.12110
25. C. Tian, M. Zheng, W. Zuo, B. Zhang, Y. Zhang, D. Zhang, Multi-stage image denoising with the wavelet transform, *Pattern Recogn.*, **134** (2023), 109050. https://doi.org/10.1016/j.patcog.2022.109050
26. Z. Chen, L. Chen, G. Cong, C. S. Jensen, Location-and keyword-based querying of geo-textual data: A survey, *VLDB J.*, **30** (2021), 603–640. https://doi.org/10.1007/s00778-021-00661-w
27. L. Jia, J. Tang, M. Li, J. You, J. Ding, Y. Chen, TWE-WSD: An effective topical word embedding based word sense disambiguation, *CAAI Trans. Intell. Technol.*, **6** (2021), 72–79. https://doi.org/10.1049/cit2.12006

AIMS Press