Compact Hilbert Indices for Multi-Dimensional Data

Chris H. Hamilton
Dalhousie University
Halifax, NS Canada
chamilton@cs.dal.ca

Andrew Rau-Chaplin
Dalhousie University
Halifax, NS Canada
arc@cs.dal.ca

Abstract

Space-filling curves, particularly Hilbert curves, have proven to be a powerful paradigm for maintaining spatial groupings of multi-dimensional data in a variety of application areas including database systems,data structures and distributed information systems. One significant limitation in the standard definition of Hilbert curves is the requirement that the grid size (i.e. the cardinality) in each dimension be the same. In the real world, not all dimensions are of equal size and the work-around of padding all dimensions to the size of the largest dimension wastes memory and disk space, while increasing the time spent manipulating and communicating these "inflated" values.

In this paper we define a new compact Hilbert index *which, maintains all the advantages of the standard Hilbert curve and permits dimension cardinalities of varying sizes. This index can be used in any application that would have previously relied on Hilbert curves but, in the case of unequal side lengths, provides a more memory efficient representation. This is particularly important in distributed applications (Parallel, P2P and Grid), in which not only is memory space saved but communication volume reduced.*

1. Introduction

At the heart of many data intensive applications is the need to store, manipulate and analyze large repositories of multi-dimensional data. Such multi-dimensional data comes in many forms [1–3, 5, 9, 18, 19, 21, 23, 25, 26, 32, 35] including spectral elements in a parallel high resolution atmospherical global circulation model [11], tissue microarray data in a co-operative Grid-based oncology system [34] or business oriented OLAP data [10].

A common challenge in all of these applications is how best to group and order the multi-dimensional data to promote efficient processing. For one dimensional data, sorting is an obvious approach as it groups data items that are close together in the key dimension. For example, if we have time-stamped transaction for a bank account, we may first sort them by time in order to then efficiently compute hourly, daily and monthly balances.

With multi-dimensional data the appropriate grouping strategy is often less obvious. We may of course pick an ordering of the dimensions, say dim_1, dim_2, ..., dim_d, and sort by it, but such an approach favours some dimensions over others. Data items that are close in dim_1 are likely to be closely grouped, while items that share values in dim_2, ..., dim_d, but not dim_1, may be very far apart. If, for example, our data items represent points in 3D space which ordering is better: x, y, z or z, x, y or one of the other four possible orderings? Note that none of these orderings of the dimensions captures the natural idea of locality, that is that points that are close together in Euclidian space (and therefore more likely to interact in any physical simulation) are grouped closely together in the resulting linear ordering.

A powerful and widely used paradigm for grouping multi-dimensional data is the use of space-filling curves [6, 8, 15, 17, 27, 31]. Space-filling curves are continuous self-similar functions that map between a one-dimensional interval and a multi-dimensional set. By convention, they are generally defined as continuous mappings from the unit interval to the unit n-dimensional hypercube. Originally formulated by Giuseppe Peano in 1890 [31] they have since found applications in a variety of fields, including mathematics [7], image processing [21], image compression [26], bandwidth reduction [30], cryptology [24], algorithms [32], scientific computing [18], parallel computing [19], geographic information systems [1] and database systems [5, 20, 23]. Such curves are good at maintaining locality in a linear ordering of multi-dimensional data. Points that are close together in the original space with respect to Euclidean distance, tend to be close together in the linear ordering defined by the curve. Of the space-filling curves, the Hilbert curve (see Figure 1) has been shown to have particularly strong locality preserving properties [27] and, as such, has been the focus of considerable research, with numerous algorithms constructed to compute it [4, 8, 17, 22, 28].

More formally, consider an n-dimensional lattice with 2^m points per dimensions,

$$\mathbb{P} = \underbrace{\mathbb{B}^m \times \cdots \mathbb{B}^m}_{n \text{ times}},$$

Proceedings of the First International Conference
on Complex, Intelligent and Software Intensive Systems (CISIS'07)
0-7695-2823-6/07 $20.00 © 2007 IEEE

where $\mathbb{B}^m = \{0,1\}^m$. A standard Hilbert index is a function

$$H : \mathbb{P} \to \mathbb{B}^{nm},$$

which maps each point to its index (interpreting $\mathbf{x} \in \mathbb{B}^{mn}$ as an integer in $\mathbb{Z}_{2^{mn}}$) on the Hilbert curve as it passes through the lattice.

Hilbert curves have been extensively used to maintain spatial groupings of multi-dimensional data in a wide variety of applications. In database systems they are used to map multi-dimensional data to linearly ordered external memory (i.e. disk drives) [16]. In data structures they are used to order multi-dimensional data to promote query efficiency [20]. And in distributed information systems they are used to *partition* multi-dimensional data in such a way that points that are close in Euclidian space are likely to be allocated to the same or neighbouring processors. The idea of using space-filling curves for partitioning has been key to applications in parallel [10], P2P [33] and grid computing [34] settings.

One limitation in the definition of Hilbert curves is the requirement that the grid size (i.e. the cardinality) in each dimension be the same (i.e. 2^m). In many applications involving points in 3D space, this is a relatively harmless assumption but in information system applications where one dimension may represent product id (cardinality $1,000,000$) while another represents gender (cardinality 2) it can be extremely wasteful. The obvious solution of padding all dimensions to the cardinality of the largest dimension wastes memory and disk space and increases processing time and communication volume when manipulating and communicating these "inflated" values.

In this paper we define a new *compact Hilbert index* which, while maintaining all of the advantages of the standard Hilbert curve, permits dimension cardinalities of varying size. More formally, consider an n-dimensional space

$$\mathbb{P}' = \mathbb{B}^{m_0} \times \cdots \times \mathbb{B}^{m_{n-1}},$$

where $m_i \in \mathbb{Z}_+$ is the *precision* of the ith dimension (there is an obvious injection $U : \mathbb{P}' \to \mathbb{P}$ that upsamples $\mathbf{p} \in \mathbb{P}'$ by prepending zeroes to each component until they have length m). Storing an element in $\mathbb{P}'$ requires $M = \sum_i m_i$ bits. However, a Hilbert index must be calculated with respect to a hypercube of precision $m = \max_i\{m_i\}$ and requires $nm \geq M$ bits of storage. Our compact Hilbert index preserves completely the ordering of H on $\mathbb{P}'$, but requires only M bits to represent. Formally, it is a mapping

$$H' : \mathbb{P}' \to \mathbb{B}^M, \tag{1}$$

such that for all $\mathbf{p}_1, \mathbf{p}_2 \in \mathbb{P}'$,

$$H(U(\mathbf{p}_1)) < H(U(\mathbf{p}_2)) \iff H'(\mathbf{p}_1) < H'(\mathbf{p}_2). \tag{2}$$

Note that the compact Hilbert index can be used in any application that would have previously relied on Hilbert curves but, in the case of unequal side lengths, provides a more memory efficient representation. This advantage is particularly important in distributed applications (parallel, P2P and grid computing), in which not only is memory space saved but communication volume is reduced.

To explore the performance of compact Hilbert indices we performed a series of experiments with both synthetic and real multi-dimensional data. In both cases, in addition to significant space savings, the use of compact Hilbert curves reduced the time required to order data in Hilbert order. For example, for a four dimensional data-set extracted from a large Apache web log, compact Hilbert indices achieved a data size reduction of 2.2 and sorting based on these indices was 4.3 times faster than the dynamic comparison routine implemented in Moore's library [28].

The remainder of this paper is organized as follows. In Section 2 we review the definition of and algorithms for computing Hilbert curves while emphasizing a geometric perspective. In Section 3 we define the notion of *compact Hilbert indices* and derive an algorithm for calculating the mapping. In Section 4 we explore the performance of compact Hilbert indices, in particular demonstrating much improved sorting times as compared to competing techniques.

2. A Geometric Approach to Hilbert Curves

In this section we describe the standard Hilbert curve from a geometric point of view and give an algorithm for finding the index on the Hilbert curve of a given point in the lattice. While motivated and derived geometrically, the resulting algorithm a variant of the *de facto* standard method presented by Butz [8] and implemented by Moore [28]. Our geometric approach highlights the source of redundancy in standard Hilbert indices and facilitates the development of compact Hilbert indices in the following section.

Consider the traditional recursive geometric construction of the two-dimensional Hilbert curve. The curve is initially defined on a 2×2 lattice with a $\sqcap$ shape as shown in Figure 1. Given an order k curve defined on a $2^k \times 2^k$ lattice we define the curve on a $2^{k+1} \times 2^{k+1}$ lattice as follows

1. Place a copy of the curve, rotated 90° counter clockwise, in the lower right cell.
2. Place a copy of the curve, rotated 90° clockwise, in the lower left cell.
3. Place a copy of the curve in each of the upper cells.
4. Connect these four disjoint curves.

The first four iterations of this method are shown in Figure 1.

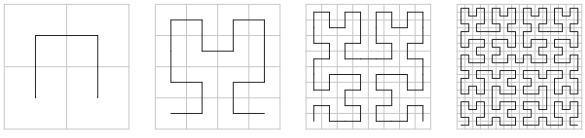


Figure 1. First four iterations of the 2D Hilbert curve, standard view.

Proceedings of the First International Conference
on Complex, Intelligent and Software Intensive Systems (CISIS'07)
0-7695-2823-6/07 $20.00 © 2007 IEEE

The basic unit of the Hilbert curve is the familiar ⊓ shape, which may be uniquely parameterized by considering the entry and exit points into the square lattice of points being walked through. Using the same approach as that taken in [4], Figure 2 illustrates the Hilbert curve where the line segments of Figure 1 have been replaced by arcs. As noted in [4] this presentation conveys more information as it indicates at some level the order in which points are visited in a given cell. The arcs show that the curve enters each cell at a given vertex, visits all the points in the cell and exits through another vertex before entering the next cell.

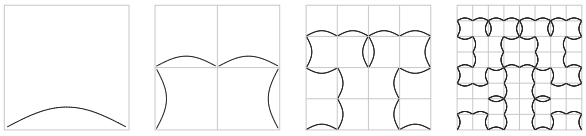


Figure 2. First four iterations of the 2D Hilbert curve, arc view.

Many algorithms for calculating Hilbert indices are based on a geometric analysis of how the curve decomposes into transformed smaller versions of itself. Nulty [29] presents a generic algorithm describing this approach with his function *SpaceKey*: 1. find the cell containing the point of interest; 2. update the key (index) value appropriately; 3. transform as necessary; and 4. continue until sufficient precision has been attained. This generic framework motivates our algorithmic approach.

Find the cell containing the point of interest Finding the cell amounts to determining whether the point lies in the upper or lower half-plane with respect to each dimension. Assuming we are working on an order m curve, a point is represented as $\mathbf{p} = [p_0, p_1] \in \mathbb{B}^m \times \mathbb{B}^m$. Determining in which half-plane the point lies with respect to the ith coordinate is equivalent to determining the truth value of $p_i < 2^{m-1}$, which is equal to the $(m-1)$th bit of p_i, $\text{bit}(p_i, m-1)$.

Update the key Given the orientation at the current resolution (uniquely defined by the entry e and exit f of the curve through the lattice), we determine the order in which each of the cells will be visited. Knowing that all points in a cell are visited before moving on to the next, the index of the cell of interest tells us whether the point of interest is visited in the first quarter of the curve, or the second and so on. Thus we may determine two bits of the Hilbert index h.

Transform as necessary Knowing the index i of the cell in which the point of interest lies, we may determine the entry and exit points of the Hilbert curve through this cell. In order to proceed, we zoom in on the cell containing the point and transform (rotate and reflect) it to the canonical orientation (entry in lower left, exit in lower right). This can be done by taking the composition of the transforms associated with our current orientation and that of the block we are zooming in on.

Continue until sufficient precision has been attained Zooming in on the cell containing our point of interest, we are now inspecting an order $m-1$ Hilbert curve through a sub-cell of our original space. We repeat this procedure for each of the remaining $m-1$ levels of precision, each time calculating a further 2 bits of the Hilbert index. At the end of the process, we have a $2m$ bit Hilbert index, isolating a single point on the length 2^{2m} curve through the $\mathbb{B}^m \times \mathbb{B}^m$ lattice.

2.1. Generalizing to Higher Dimensions

The described approach yields an algorithm for the calculation of two-dimensional Hilbert indices. In order to generalize it to higher dimensions, we need to identify the properties of the Hilbert curve we wish to generalize. The first observation relates to the order of the curve through cells. In two dimensions, successive cells are immediate neighbors along exactly one dimension. Given a 2 bit labeling for each of the cells, this means that in labels of successive sub-cells *exactly* one bit will change. This is simply a *Gray Code* [13] over 2 bit integers. In n dimensions, we have 2^n cells each labeled with an n-bit string and we may use the n-bit Gray Code to impose an ordering on the cells.

A Gray Code may be interpreted as a Hamiltonian circuit through the vertices of a hypercube in n dimensions. This implies that the first (entry) and last (exit) points are also immediate neighbors. Thus, we may determine the orientation of a given cell i by considering the entry $\mathrm{e}(i)$ into the cell and the dimension $0 \leq \mathrm{d}(i) < n$ along which the exit point is its neighbor. Having chosen the Gray Code order as the ordering through the cells, consistent orientations of each cell have to be determined such that the exit vertex of a cell is immediately adjacent to the entry vertex of the next cell. In [14], a closed form is derived for $\mathrm{e}(i), \mathrm{f}(i)$ and $\mathrm{d}(i)$. An order 2 three-dimensional Hilbert curve and the associated arc representation may be found in Figure 3.

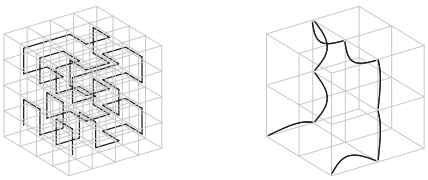


Figure 3. Standard and arc views of the order 2 three-dimensional Hilbert curve.

A full analysis of the rotations and reflections involved in the calculation of the Hilbert index shows that they may all be expressed very naturally in base 2 arithmetic. In fact, reflection may be viewed as the exclusive-or ($\veebar$) operation

Proceedings of the First International Conference
on Complex, Intelligent and Software Intensive Systems (CISIS'07)
0-7695-2823-6/07 $20.00 © 2007 IEEE

and rotation as a bitwise rotation ($\circlearrowleft$) operation. These simplifications lead directly to the formulation of Algorithm 1. For full details, including proofs and inverse algorithms, refer to [14].

Algorithm 1 HILBERTINDEX$(n, m, \mathbf{p})$

Calculates the Hilbert index of a point.

Input: $n, m \in \mathbb{Z}_+$ and a point $\mathbf{p} \in \mathbb{P}$.
Output: $h \in \mathbb{B}^{nm}$, the Hilbert index of the point $\mathbf{p} \in \mathbb{P}$.

1: $(h, e, d) \leftarrow (0, 0, 0)$
2: **for** $i = m - 1$ to 0 **do**
3: $\quad l \leftarrow [\text{bit}(p_{n-1}, i) \cdots \text{bit}(p_0, i)]_{[2]}$ `// Get cell label`
4: $\quad t \leftarrow (l \veebar e) \circlearrowleft d$ `// Transform to canonical orientation`
5: $\quad w = \text{gc}^{-1}(t)$ `// Determine cell index in gc order`
6: $\quad h \leftarrow (h \triangleleft n) \vee w$ `// Add n bits to Hilbert index`
7: $\quad e \leftarrow e \veebar (\text{e}(w) \circlearrowleft d)$ `// Compose transforms`
8: $\quad d \leftarrow d + \text{d}(w) + 1 \bmod n$
9: **end for**

Algorithm 1 is clearly visible as falling under the *SpaceKey* framework of [29]. In contrast, Butz's algorithm merges the transformation (line 4) and composition (line 7) into a compound operation and moves the inverse Gray Code operation (line 5) outside of the loop, leading to a more terse implementation with less intermediate variables. However, having each of the *SpaceKey* steps separated out facilitates the development of compact Hilbert indices.

3. Compact Hilbert Indices

As discussed in Section 1 it is desirable to have a mapping that preserves the relative ordering of the Hilbert curve but does not require additional space to represent. A simple method to construct such a mapping is to walk through all the points in $\mathbb{P}'$, calculate their Hilbert indices and sort them based on these values. Then, assign to each point $\mathbf{p}$ its rank in this sorted list as an index. Trivially, this index has the same ordering as the Hilbert ordering over $\mathbb{P}'$ and it requires only $M = \sum_i m_i$ bits to represent. However, in order to generate the index in this manner we must first enumerate the entire space, a prohibitive cost. The key to calculating this index directly, referred to as the *compact Hilbert index*, lies in an observation of how bits from the point $\mathbf{p}$ travel through Algorithm 1 and contribute to the Hilbert index.

3.1. An Observation

We inspect line 3 of Algorithm 1 which calculates the location l of the point $\mathbf{p}$ as

$$l = [\text{bit}(p_{n-1}, i) \cdots \text{bit}(p_0, i)]_{[2]}.$$

Due to the varying precisions of each coordinate we know that for any point $\mathbf{p} \in \mathbb{P}'$, $\text{bit}(p_j, i) = 0$ when $i \geq m_j$. Thus at any given iteration i, some subset of the n bits of l may be fixed and known to be zero. These bits do not provide any information to the calculation, yet they are still used to calculate a full n bits of the index h. Following these redundant bits through lines 4-5 shows how we can discard them while still preserving the ordering of the points in $\mathbb{P}'$ as visited by the Hilbert curve over $\mathbb{P}$.

Let $\mathcal{A}_i = \{j : m_j > i, 0 \leq j < n\}$ be the set of "active" dimensions at iteration i. Consider the calculation of t, the transformed location, on line 4 of Algorithm 1:

$$t = (l \veebar e) \circlearrowleft d.$$

Since every inactive bit of l is zero valued, then the bits of $l \veebar e$ at these positions will simply take on the value of the corresponding bits of e. Thus the only bits of $l \veebar e$ whose values are "free" are those in $\mathcal{A}_i$. The rotation operator only shuffles the bits of $l \veebar e$ in a simple manner. Let $\mathcal{F}_i$ be the set of free bits of t at iteration i; that is, those bits whose values are affected by l and in turn $\mathbf{p}$. It is easy to see that

$$\begin{aligned} \mathcal{F}_i &= \{j : j + d \bmod n \in A_i\} \\ &= \{j : m_{(j+d \bmod n)} > i, 0 \leq j < n\}. \end{aligned}$$

Since $|\mathcal{F}_i| = |\mathcal{A}_i|$, then we see that both l and t may only be one of $2^{|\mathcal{F}_i|}$ unique values. Additionally, we know that the Gray Code, and hence its inverse, is a bijective operator over $\mathbb{B}^n$ (see [14] for a proof). Thus the final value w may also only be one of $2^{|\mathcal{F}_i|}$ distinct values.

Let r be the rank of w with respect to all possible values w may take on at a given iteration i of the algorithm. Then r is a $|\mathcal{F}_i|$-bit integer and satisfies

$$r_1 < r_2 \iff w_1 < w_2. \tag{3}$$

Instead of appending w to our partially calculated index h, we may append the rank r. By Equation 3 we see that for any $\mathbf{p}_1, \mathbf{p}_2 \in \mathbb{P}'$, the following holds with respect to modified indices, h_1 and h_2, constructed from rank values:

$$h_1 < h_2 \iff H(U(\mathbf{p}_1)) < H(U(\mathbf{p}_2))$$

Additionally, we see that the modified indices have a bit length of

$$\begin{aligned} \sum_{i=0}^{m-1} |\mathcal{F}_i| &= \sum_{i=1}^{m-1} |\{j : m_j > i, 0 \leq j < n\}| \\ &= \sum_{j=0}^{n-1} |\{i : m_j > i, 0 \leq i < m\}| = \sum_{j=0}^{n-1} m_j = M. \end{aligned}$$

Thus such an index satisfies Equations 1 and 2, as desired. It remains only to show how to calculate the rank r of a value w given $\mathcal{F}$. We first consider an example.

Example 3.1 (Gray Code Rank) *We consider the values of t, w and r for $n = 6$, $(e \circlearrowleft d) = [001000]_{[2]}$ and 3 free bits. In the following table the free bits of t have been underlined while the ranks r have been calculated by inspection over the set of all w values.*

t	8	10	12	14
w	15	12	8	11
r	3	2	0	1
$[t]_{[2]}$	0$\underline{01}$000	0$\underline{01}$010	0$\underline{01}$100	0$\underline{01}$110
$[w]_{[2]}$	0$\underline{01}$111	0$\underline{01}$100	0$\underline{01}$000	0$\underline{01}$011
$[r]_{[2]}$	011	010	000	001
t	20	26	28	30
w	16	19	23	20
r	4	5	7	6
$[t]_{[2]}$	0$\underline{11}$000	0$\underline{11}$010	0$\underline{11}$100	0$\underline{11}$110
$[w]_{[2]}$	0$\underline{10}$000	0$\underline{10}$011	0$\underline{10}$111	0$\underline{10}$100
$[r]_{[2]}$	100	101	111	110

As can be seen, the rank r of w can be constructed by extracting the free bits $f \in \mathcal{F}$ from the Gray Code index w. We formalize this in Lemma 3.4 and Theorem 3.5, first stating without proof a few necessary lemmas.

Proceedings of the First International Conference
on Complex, Intelligent and Software Intensive Systems (CISIS'07)
0-7695-2823-6/07 $20.00 © 2007 IEEE

Lemma 3.2 (Gray Code, Theorem 2.1 of [14]) *Consider a non-negative integer $w \in \mathbb{B}^m$. Let $t = \mathrm{gc}(w)$. Then it follows that $t = w \veebar (w \triangleright 1)$, or equivalently,* $\mathrm{bit}(t, j) = \mathrm{bit}(w, j) + \mathrm{bit}(w, j+1) \bmod 2$.

Lemma 3.3 (Gray Code Inverse, Theorem 2.2 of [14]) *Consider a non-negative integer $t \in \mathbb{B}^m$. Let $w = \mathrm{gc}^{-1}(t)$. Then it follows that*

$$\mathrm{bit}(w, j) = \sum_{k=j}^{m-1} \mathrm{bit}(t, k).$$

Lemma 3.4 (Principal Bits) *Given e, d and i, let*

$$\mathcal{F} = \{j : m_{(j+d \bmod n)} > i, 0 \leq j < n\}.$$

Let $\mathcal{T}$ be the set of 2^k distinct values that may differ from $(e \circlearrowright d)$ only at the $k = |\mathcal{F}|$ bits $j \in \mathcal{F}$. Consider $a \neq b \in \mathcal{T}$. Let l be the index of the most significant bit of a and b that does not match; in other words, $l = \max\{k : \mathrm{bit}(a, k) \neq \mathrm{bit}(b, k)\}$. It follows that $l \in \mathcal{F}$.

Proof. Define a mask μ as the n-bit integer such that

$$\mathrm{bit}(\mu, j) = \begin{cases} 0, & j \notin \mathcal{F}, \\ 1, & j \in \mathcal{F}. \end{cases}$$

The mask μ is created such that only bits in free positions are one valued. Since $t \in \mathcal{T}$ may only differ from $(e \circlearrowright d)$ at the bits $j \in \mathcal{F}$, we may rewrite

$$\mathcal{T} = \{t : t \wedge \mu = (e \circlearrowright d) \wedge \mu, t \in \mathbb{B}^n\}.$$

By Lemma 3.3 it follows that

$$\mathrm{bit}(a, l) = \sum_{l \leq k < n} \mathrm{bit}(\mathrm{gc}(a), k) \bmod 2.$$

Knowing $\mathrm{bit}(a, k) = \mathrm{bit}(b, k)$ for $k > l$, Lemma 3.2 implies $\mathrm{bit}(\mathrm{gc}(a), k) = \mathrm{bit}(\mathrm{gc}(b), k)$ for $j > l$. Thus:

$$\begin{array}{rcl} \begin{array}{c}\mathrm{bit}(a,l)\\ +\\ \mathrm{bit}(b,l)\end{array} & = & \sum_{l \leq k < n} \big(\mathrm{bit}(\mathrm{gc}(a), k) + \mathrm{bit}(\mathrm{gc}(b), k)\big) \bmod 2 \\ & = & \mathrm{bit}(\mathrm{gc}(a), l) + \mathrm{bit}(\mathrm{gc}(b), l). \end{array}$$

Suppose $l \notin \mathcal{F}$. Then it follows that $\mathrm{bit}(\mathrm{gc}(a), l) = \mathrm{bit}(\mathrm{gc}(b), l) = \mathrm{bit}(e \circlearrowright d, l)$ and therefore $\mathrm{bit}(a, l) = \mathrm{bit}(b, l)$, a contradiction. Hence, $l \in \mathcal{F}$. ■

Theorem 3.5 (Gray Code Rank) *Let $\mathcal{F}, \mathcal{T}$ and μ be as in Lemma 3.4. Define the Gray Code Rank as*

$$\mathrm{gcr}(w) = [\mathrm{bit}(w, f_{k-1}), \ldots, \mathrm{bit}(w, f_0)]_{[2]},$$

where $\mathcal{F} = \{f_0 < \cdots < f_{k-1}\}$ and $w = \mathrm{gc}^{-1}(t)$ for some $t \in \mathcal{T}$. Then for all $t_1, t_2 \in \mathcal{T}$ it follows that

$$\mathrm{gc}^{-1}(t_1) < \mathrm{gc}^{-1}(t_2) \iff \mathrm{gcr}(\mathrm{gc}^{-1}(t_1)) < \mathrm{gcr}(\mathrm{gc}^{-1}(t_2)).$$

Proof. Lemma 3.4 states that the most significant differing bit between $\mathrm{gc}^{-1}(t_1)$ and $\mathrm{gc}^{-1}(t_2)$ must be a free bit. In other words, the only bits necessary to compare the relative order of these two values are precisely the bits of index $f \in \mathcal{F}$. Thus, if we remove the constrained bits from $\mathrm{gc}^{-1}(t_1)$ and $\mathrm{gc}^{-1}(t_2)$ keeping only the free bits, we are left with two $|\mathcal{F}|$ bit values which preserve the ordering of $\mathrm{gc}^{-1}(t_1)$ and $\mathrm{gc}^{-1}(t_2)$. This corresponds exactly to the values $\mathrm{gcr}(\mathrm{gc}^{-1}(t_1))$ and $\mathrm{gcr}(\mathrm{gc}^{-1}(t_2))$. ■

The results of this section and particularly Theorem 3.5 give us the last tools required to create an algorithm to compute compact Hilbert indices. Using Algorithm 1 as a starting point, Algorithms 2, 3 and 4 allow the computation of the mapping H'.

Algorithm 2 EXTRACTMASK$(n, m_0, \ldots, m_{n-1}, i, d)$

Extracts a mask μ indicating which bits of t are free at iteration i of the COMPHILBERTINDEX algorithm.

Input: $n, m_0, \ldots, m_{n-1} \in \mathbb{Z}_+$, $i \in \mathbb{Z}_m$ and $d \in \mathbb{Z}_n$.
Output: μ.

```
μ ← 0
for j = n − 1 to 0 do
    μ ← μ ◁ 1
    if m_(j+d mod n) > i then
        μ ← μ ∨ 1
    end if
end for
```

Algorithm 3 GRAYCODERANK(n, μ, w)

Returns the Gray Code rank of w with respect to μ.

Input: $n \in \mathbb{Z}_+$, $\mu, w \in \mathbb{B}^n$.
Output: w.

```
r ← 0
for j = n − 1 to 0 do
    if bit(μ, j) = 1 then
        r ← (r ◁ 1) ∨ bit(w, j)
    end if
end for
```

Algorithm 4 COMPHILBERTINDEX$(n, m_0, \ldots, m_{n-1}, \mathbf{p})$

Calculates the compact Hilbert index of a point.

Input: $n, m_0, \ldots, m_{n-1} \in \mathbb{Z}_+$ and $\mathbf{p} \in \mathbb{P}'$.
Output: $h_c \in \mathbb{B}^M$.

```
(h_c, e, d) ← (0, 0, 0)
m ← max_i{m_i}
for i = m − 1 to 0 do
    μ ← EXTRACTMASK(n, m_0, ..., m_{n−1}, i, d)
    l ← [bit(p_{n−1}, i) ··· bit(p_0, i)]_[2]
    t ← (l ⊻ e) ↻ d
    w ← gc^{−1}(t)
    r ← GRAYCODERANK(n, μ, w)
    h_c ← (h_c ◁ ||μ||) ∨ r
    e ← e ⊻ (e(w) ↻ d)
    d ← d + d(w) + 1 mod n
end for
```

Inspection shows that each of EXTRACTMASK and GRAYCODERANK have $O(n)$ time complexity. Similarly, Lemma 3.3 shows that calculating $\mathrm{gc}^{-1}(t)$ requires at most $O(n)$ (in fact, it is $O(\log n)$). Given that each of $\mathrm{e}(w), \mathrm{d}(w)$ and $\circlearrowright$ may be implemented in at most $O(n)$ complexity, we see that both Algorithms 1 and 4 have a net complexity of $O(nm)$. Specifically, this shows that compact Hilbert indices are at most a constant factor more expensive to compute than regular Hilbert indices.

Proceedings of the First International Conference
on Complex, Intelligent and Software Intensive Systems (CISIS'07)
0-7695-2823-6/07 $20.00 © 2007 IEEE

4. Experimental Results

To quantify the performance of our algorithms we implemented routines for mapping to and from both regular and compact Hilbert indices. The algorithms are written in C++ and seamlessly handle arbitrary precision[1]. Our Hilbert curve algorithms were then compared to Moore's [28] *de facto* standard implementation of Butz's [8] algorithms for various precisions and dimensions (up to $nm \leq 64$, the maximum supported by Moore's code) on both artificial and real data. The running times of our compact Hilbert indices were then compared to those of regular Hilbert indices over these same and larger data-sets. Finally, we examine the effect of using compact Hilbert indices in applications where regular Hilbert indices are currently used. All experiments were performed on a commodity Dual Intel Xeon 3.06GHz based computer with 2GB of memory. All quoted times are wall times.

4.1. Data-sets

The WEBLOG data-set consists of the log files of an Apache web server, taken over a 139 day period from August to September of 2004. A four-dimensional data-set was extracted from the over 154 million rows of log data, as summarized in Table 1.

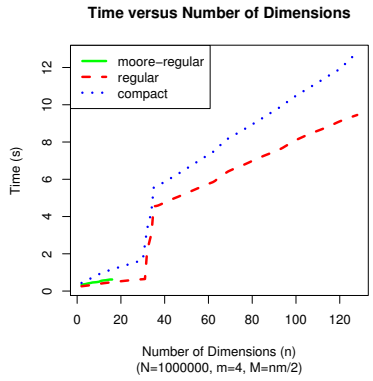
Table 1. Summary of the WEBLOG data-set.

i	Quantity	Cardinality	Precision (m_i)
0	IP address	834,406	20
1	day of access	139	8
2	hour of access	24	5
3	HTTP return code	16	4
Dimensionality		$n = 4$	
Maximum precision		$m = 20$	
Size of Hilbert index		$nm = 80$	
Size of compact Hilbert index		$M = 37$	
Hilbert index expansion factor		$nm/M = 2.2$	
Number of data points		$N = 7,709,286$	

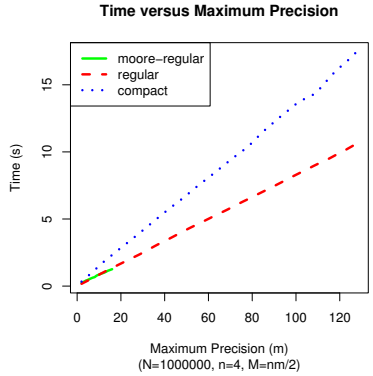
4.2. Performance

In order to characterize the performance of our algorithms we compared them against Moore's code over randomly generated data-sets and varying parameters for N, m, n and M. For the purposes of compact Hilbert indices, precisions m_i were chosen in a monotonically decreasing fashion such that $M = nm/2$. Figure 4 shows the basic results. The jump visible at $n = 32$ in Figure 4(a) is due to the code switching to multiple precision representations of n-bit intermediate variables. In general, our regular Hilbert curve implementation slightly outperforms that of Moore. When $n \leq 32$ the overhead associated with compact Hilbert indices is as much as 2.5 times or 150%. However, as both n and m increase this reduces to a more reasonable 40%. Although the compact Hilbert indices take slightly longer to compute, they are smaller than full Hilbert indices allowing data points to be replaced with compact Hilbert indices in-place.

[1] Source available at http://cs.dal.ca/~chamilto/hilbert/.



(a)



(b)

Figure 4. Performance comparison over randomly created data-sets. (a) Time to calculate N Hilbert indices with $m = 4$ as n varies. (b) Time to calculate N Hilbert indices with $n = 4$ as m varies.

4.3. Sorting by Hilbert Index

As discussed in Section 1, Hilbert curves are often used to order or partition multi-dimensional data. Thus it becomes necessary to sort points by their Hilbert indices. The simplest approach is to simply calculate the Hilbert index for each point, and use this value in sorting. However, often these indices are larger than the points they represent resulting in an increased storage cost. Given the large nature of the data-sets being sorted, it is often critical that the sort be in-place. Moore's solution to this problem was to create a dynamic comparison routine which simultaneously calculates the Hilbert index of both points being compared. It calculates the indices only to the precision required to determine the relative order of the two points. This approach has the benefit that the Hilbert indices are never explicitly stored, but suffers from the problem that they are recalcu-

lated at every comparison.

Dynamic Hilbert Sorting Suppose a comparison requires examining the first b bits of the Hilbert indices of two points in order to distinguish them. Since each bit costs $O(1)$ to calculate, this incurs a cost of $O(b)$. Fill et al [12] explored the average bit-cost per comparison assuming quick-sort is being used. They derive an expected $O(\log N)$ bits per comparison, which implies a total bit complexity of $O(N \log^2 N)$. Thus, using a quick-sort based algorithm we can expect a total dynamic Hilbert index sorting run-time of $O(N \log^2 N)$. Although this particular analysis is valid only for the quick-sort algorithm, it is thought this bound holds for the general problem of sorting[2].

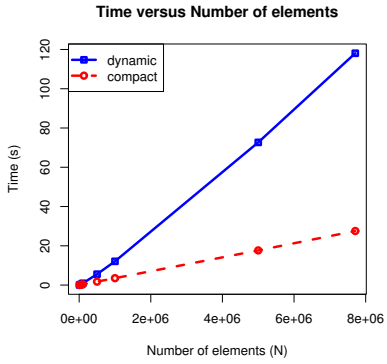
Compact Hilbert Sorting As a competing approach we consider sorting using compact Hilbert indices. Since compact Hilbert indices are the same size as the data from which they are calculated, we first replace the data points with their associated compact Hilbert indices at a net cost of $O(Nnm)$. We then sort these elements before converting back to the original data points. The net cost of this sort is $O(N(\log N + nm))$. As long as $nm < \log^2 N$, such an approach can be expected to be asymptotically faster than dynamic Hilbert sorting.

Figure 5a shows the results of sorting the `WEBLOG` dataset using both dynamic Hilbert indices and compact Hilbert indices. As predicted, compact Hilbert sorting proved to be much more efficient. As shown in Figure 5b, for as little as 100K data items a speedup of 2 was observed. By 1M data items that speedup had grown to a factor of 3.4. Speedup continued to increase beyond this point until reaching a factor of 4.3 on the whole data set. Note that in distributed applications that order and partition data using Hilbert curves, such as [10, 33, 34], the benefits of using compact Hilbert curves would be even more pronounced. The use of compact Hilbert curves would not only result in the memory and computation savings illustrated in Figure 5 but would also reduce the size of the data to be communicated and therefore increase the overall communication speed.

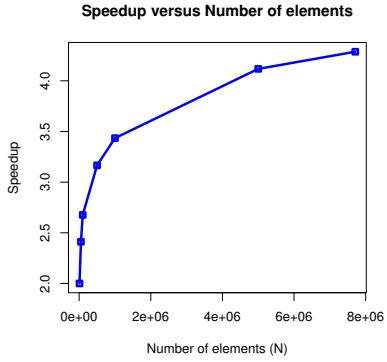
5. Conclusion

Due their wide variety of uses and simple elegance, space-filling curves have been researched continuously since their discovery, finding many applications Motivated by the lack of intuition in the near-ubiquitous Butz [8] algorithms for Hilbert curves as implemented by Moore [28], we have reconstructed these algorithms from a geometric point of view. Based on this formulation we have then described a compact Hilbert curve which captures the ordering

[2]Under the constraint that in order to compare a bit, we must first have compared all bits more significant than it; if we have random bit access a radix sort can generally do better. Hilbert indices are calculated incrementally, precluding random bit access.



(a)



(b)

Figure 5. Comparing dynamic Hilbert sorting and compact Hilbert sorting using the `WEBLOG` data-set. The `compact` curve includes the cost of converting both to and from compact Hilbert indices. (a) Wall times. (b) Relative speed-up.

properties of the regular Hilbert curve but without the associated inefficiency in representation for spaces with unequal side lengths. Finally, we developed algorithms for dealing with compact Hilbert indices and demonstrated their performance and utility in real-world applications. Although these compact Hilbert indices are somewhat more computationally expensive to derive, they result in significant space savings and, in the critical operation of sorting by Hilbert indices, they result in a considerable time saving. For example, for a typical four dimensional web log data set, compact Hilbert indices achieved a data size reduction of 2.2 and a sorting speedup of 4.3 over the widely used dynamic Hilbert sort method. It is our hope that the compact Hilbert indices introduced in this paper will find uses in information systems and other applications in which multi-dimensional spaces have dimensions of unequal cardinalities.

References

[1] D. J. Abel and D. M. Mark. A comparative analysis of some two-dimensional orderings. *Int. J. of Geo. Inf. Syst.*, 4(1):21–31, Jan 1990.

[2] C. Alpert and A. Kahng. Multi-way partitioning via space-filling curves and dynamic programming. In *31st Conf. on Design Automation*, pp. 652–657, Jun 6-10 1994.

[3] J. J. Bartholdi III. A routing system based on spacefilling curves. http://www2.isye.gatech.edu/˜jjb/mow/mow.pdf, Apr 1995.

[4] J. J. Bartholdi III and P. Goldsman. Vertex-labeling algorithms for the Hilbert spacefilling curve. *Software–Practice and Experience*, 31(5):395–408, May 2001.

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

[6] G. Breinholt and C. Schierz. Algorithm 781: Generating Hilberts space-filling curve by recursion. *ACM Trans. on Math. Software*, 24(2):184–189, Jun 1998.

[7] A. R. Butz. Convergence with Hilbert's space-filling curve. *J. of Comp. and Syst. Sciences*, 3(2):128–146, May 1969.

[8] A. R. Butz. Alternative algorithm for Hilbert's space-filling curve. *IEEE Trans. on Comp.*, pp. 424–426, Apr 1971.

[9] S. Chatterjee, A. R. Lebeck, P. K. Patnala, and M. Thottethodi. Recursive array layouts and fast parallel matrix multiplication. In *11th ACM Symp. on Par. Alg. and Arch.*, pp. 222–231, Jun 27-30 1999.

[10] F. Dehne, T. Eavis, and A. Rau-Chaplin. Parallel multi-dimensional rolap indexing. In *IEEE Int. Symp. on Cluster Comp. and the Grid*, pp. 86–93, 2003.

[11] J. M. Dennis, H. M. Tufo, and R. D. Loft. Partitioning the cubed-sphere for bluegene/l. In *11th SIAM Conf. on Par. Proc. for Sci. Comp.* SIAM Press, 2004.

[12] J. A. Fill and S. Janson. The number of bit comparisons used by quicksort: an average-case analysis. In *15th ACM-SIAM Symp. on Discrete Alg.*, pp. 300–307, 11-13 Jan 2004.

[13] F. Gray. Pulse code communication. US Patent Number 2,632,058, Mar 17 1953.

[14] C. Hamilton. Compact Hilbert indices. Technical Report CS-2006-07, Dalhousie University, Faculty of Computer Science, Jul 2006.

[15] D. Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück. *Math. Annalen*, 38:459–460, 1891.

[16] H. V. Jagadish. Linear clustering of objects with multiple attributes. In *ACM Int. Conf. on Management of Data*, pp. 332–342, 1990.

[17] G. Jin and J. M. Mellor-Crummey. SFCGen: A framework for efficient generation of multi-dimensional space-filling curves by recursion. *ACM Trans. on Math. Software*, 31(1):120–148, Mar 2005.

[18] G. Jin, J. M. Mellor-Crummey, and R. J. Fowler. Increasing temporal locality with skewing and recursive blocking. In *ACM/IEEE Conf. on Supercomp.*, p. 43, Nov 10-16 2001.

[19] M. Kaddoura, C.-W. Ou, and S. Ranka. Partitioning unstructured computational graphs for nonuniform and adaptive environments. *IEEE Par. and Dist. Tech.: Syst. and Tech.*, 3(3):63–69, Sep 1995.

[20] I. Kamel and C. Faloutsos. Hilbert R-tree: An improved R-tree using fractals. In *12th Int. Conf. on Very Large Databases*, pp. 500–509, Sep 1994.

[21] C.-H. Lamaque and F. Robert. Image analysis using space-filling curves and 1d wavelet bases. *Pattern Recognition*, 29(8):1309–1322, Aug 1996.

[22] J. K. Lawder. Calculations of mappings between one and n-dimensional values using the Hilbert space-filling curve. Technical Report JL1/00, Birkbeck College, University of London, Aug 2000.

[23] J. K. Lawder and P. J. H. King. Querying multi-dimensional data indexed using the Hilbert space-filling curve. *SIGMOD Record*, 30(1):19–24, Mar 2001.

[24] Y. Matia and A. Shamir. A video scrambling technique based on space filling curves. In *Advances in Crypto.*, pp. 398–417, Aug 16-20 1987.

[25] J. Mellor-Crummey, D. Whalley, and K. Kennedy. Improving memory hierarchy performance for irregular applications using data and computation reorderings. *Int. J. of Par. Prog.*, 29(3):217–247, 2001.

[26] B. Moghaddam, K. Hintz, and C. Stewart. Space-filling curves for image compression. In *1st SPIE Conf. on Auto. Obj. Recogn.*, volume 1471, pp. 414–421, Apr 1-5 1991.

[27] B. Moon, H. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of the Hilbert space-filling curve. *Knowledge and Data Eng.*, 13(1):124–141, Jan 2001.

[28] D. Moore. Fast Hilbert curve generation, sorting, and range queries. http://web.archive.org/web/20050212162158/http://www.caam.rice.edu/˜dougm/twiddle/Hilbert/, 1999.

[29] W. G. Nulty. *Geometric Searching with Spacefilling Curves*. PhD thesis, Georgia Institute of Technology, Atlanta, GA, 1993.

[30] E. A. Patrick, D. R. Anderson, and F. Bechtel. Mapping multidimensional space to one dimension for computer output display. *IEEE Trans. on Comp.*, 17(10):949–953, Oct 1968.

[31] G. Peano. Sur une courbe, qui remplit toute une aire plane. *Math. Annalen*, 36:157–160, 1890.

[32] L. K. Platzman and J. J. Bartholdi III. Spacefilling curves and the planar travelling salesman problem. *J. of the ACM*, 36(4):719–737, Oct 1989.

[33] C. Schmidt and M. Parashar. Enabling flexible queries with guarantees in P2P systems. *IEEE Internet Comp.*, 08(3):19–26, 2004.

[34] C. Schmidt, M. Parashar, W. Chen, and D. J. Foran. Engineering a peer-to-peer collaboratory for tissue microarray research. In *2nd Int. Workshop on Challenges of Large Appl. in Dist. Env.* IEEE Comp. Society, 2004.

[35] Y. Zhang and R. E. Webber. Space diffusion: An improved parallel halftoning technique using space-filling curves. In *20th Conf. on Comp. Graphics and Interactive Techniques*, pp. 305–312, Aug 2-6 1993.

Proceedings of the First International Conference
on Complex, Intelligent and Software Intensive Systems (CISIS'07)
0-7695-2823-6/07 $20.00 © 2007 IEEE