12-2012

Approximate covering detection among content-based subscriptions using space filling curves

Zhenhui Shen
Iowa State University

Srikanta Tirthapura
Iowa State University, snt@iastate.edu

Approximate covering detection among content-based subscriptions using space filling curves

Abstract
Subscription covering is an optimization that reduces the number of subscriptions propagated, and hence, the size of routing tables, in a content-based publish-subscribe system. However, detecting covering relationships among subscriptions can be a costly computational task; this cost can potentially reduce the utility of covering as an optimization. We introduce an alternate approach called approximate subscription covering, that can provide much of the benefits of subscription covering at a fraction of the cost. By forgoing an exhaustive search for covering subscriptions in favor of an approximate search, it is shown that the time complexity of covering detection can be dramatically reduced. The tradeoff between efficiency of covering detection and the approximation error is demonstrated through the analysis of indexes for multi-attribute subscriptions based on space filling curves.

Keywords
Publish-subscribe, Subscription covering, Space filling curve

Comments
This is a manuscript of an article published as Shen, Zhenhui, and Srikanta Tirthapura. "Approximate covering detection among content-based subscriptions using space filling curves." *Journal of Parallel and Distributed Computing* 72, no. 12 (2012): 1591-1602. DOI: 10.1016/j.jpdc.2012.09.002. Posted with permission.

Approximate Covering Detection among Content-Based Subscriptions Using Space Filling Curves

Zhenhui Shen Srikanta Tirthapura*

Abstract

We consider a problem that arises during the propagation of subscriptions in a content-based publish-subscribe system. Subscription covering is a promising optimization that reduces the number of subscriptions propagated, and hence the size of routing tables in a content-based publish-subscribe system. However, detecting covering relationships among subscriptions can be an expensive computational task that potentially reduces the utility of covering as an optimization.

We introduce an alternate approach *approximate subscription covering*, which provide much of the benefits of subscription covering at a fraction of its cost. By forgoing an exhaustive search for covering subscriptions in favor of an approximate search, it is shown that the time complexity of covering detection can be dramatically reduced. The trade off between efficiency of covering detection and the approximation error is demonstrated through the analysis of indexes for multi-attribute subscriptions based on space filling curves.

1 Introduction

Content-based publish-subscribe systems offer an expressive middleware to distributed applications. These systems route messages from senders (publishers) to receivers (subscribers) based on the message content, rather than on a fixed destination address, as in conventional IP routing. In order to express their personal interests, subscribers provide *subscriptions* to the system, which are predicates on the message content. A subscriber is delivered only those messages which match

*Dept. of Electrical and Computer Engg., Iowa State University, 2215 Coover Hall, Ames, IA 50010, USA, Email: {zhshen,snt}@iastate.edu

its subscriptions. For example, the event [stock = IBM, volume = 1000, current = 88] must be delivered to a client which has issued the subscription [stock = IBM, volume > 500, current < 95]. A content-based routing middleware greatly simplifies the design of distributed applications. Since a centralized implementation of publish-subscribe may not scale to large networks, many existing systems like Gryphon [ZSB04], Siena [CRW04], JEDI [CNF01] and REBECA [MFB02] use a distributed network of routers to implement publish-subscribe.

Subscription propagation is a key component in any distributed publish-subscribe system. A subscription registered at one router has to be propagated within the network so that events published at remote routers can be delivered back. One approach to propagating subscriptions is to flood each subscription to all routers in the network. However, such a naive approach can greatly increase network traffic as well as the size of routing tables, making event forwarding less efficient. A more efficient approach (that has been proposed by multiple groups [LHJ05, CRW04, TE04, SAT05]) is to take advantage of *covering* relationships among subscriptions to reduce the number of subscriptions propagated in the system. For a subscription s, let $N(s)$ denote the set of all messages that match s. Let s_1 and s_2 be two arbitrary subscriptions. We say that s_1 *covers* s_2, iff $N(s_1) \supseteq N(s_2)$. If a newly arrived subscription s_2 is covered by an existing subscription s_1, then there is no need to forward s_2, since all messages that are matching s_2 are already being received. Repeating this process at every router in the network can lead to a significant reduction in the size of the routing tables at the nodes, and eventually lead to shorter delivery latencies. Subscription covering has been implemented in existing systems, including Siena [CRW04], JEDI [CNF01] and REBECA [MFB02].

However, identifying covering relationships among subscriptions is itself a hard combinatorial problem, for which there are no worst-case time efficient solutions known. Consider a system where each message is composed of multiple numeric attributes, and each subscription is a conjunction of range constraints over the attributes. Suppose a router has already received a set of subscriptions, S. Given an arriving subscription s, the problem of deciding whether or not there is a subscription in S that covers s can be shown to be equivalent to the *point dominance* problem in high-dimensional space. Point dominance is well studied in computational geometry and strong lower bounds are known for its time and space complexity [Cha90b, Cha90a], showing that there does not exist a worst-case time efficient solution. This is called the *curse of dimensionality* –

the cost of indexing geometric objects in high dimensions often increases exponentially with the number of dimensions. While there exist worst-case efficient data structures for point dominance in one and two dimensional spaces, the same is not true for higher dimensions.

Approximate Subscription Covering. We observe that in our publish-subscribe application, there is no need to search for covering subscriptions exhaustively every time. Covering is only an optimization; the system will continue to work correctly if covering relationships go undetected. However, if routers continue to forward subscriptions that are covered, the system could soon degrade in performance. Thus, we have two extremes, neither of them very desirable – one is to ignore covering completely and the other is to follow it exactly all the time. We propose a middle ground, *approximate* covering detection. When a new subscription arrives, the set of existing subscriptions at a router is partially searched for a covering subscription, and if none is found, then the new subscription is forwarded. We precisely quantify the fraction of the space of covering subscriptions that is searched. Our main result is that this middle ground is quite attractive. *It is possible to search most of the solution space consisting of covering subscriptions at a fraction of the cost it takes to exhaustively search for covering subscriptions.*

We design indexes for approximate covering based on *space-filling curves*. A space filling curve (referred to as "SFC" henceforth) is a proximity preserving mapping from a high dimensional space to a single dimension. SFCs are one of the most popular techniques for indexing high-dimensional data, and have been widely used for tasks such as nearest neighbor search and range queries in high-dimensional spaces [Fal88], data partitioning in parallel computers [LAB+02], and in scientific computing [WS93, NKV99]. However, as explained above, SFCs (or any other data structure, for that matter) do not provide worst-case time efficient solutions to point dominance, and hence to subscription covering. We show however, that they can be used to answer approximate point dominance queries much more efficiently.

1.1 Subscription Covering through Point Dominance

We consider a publish-subscribe system where each message has β numerical attributes, and each subscription is a conjunction of range constraints, with one constraint per attribute. A message can be treated as a point in β dimensional space, and a subscription can be treated as a β dimensional rectangle that matches all messages whose corresponding points lie inside the rectangle.

Let S denote the set of subscriptions that have already arrived at the router. Given an incoming subscription s, the problem of finding whether or not it is covered by an existing subscription in S is equivalent to the problem of finding an existing rectangle that encloses the incoming rectangle. We apply the following well-known transformation (Edelsbrunner and Overmars[EO82]) to convert this into an equivalent *point dominance* problem in 2β-dimensional space. A β-dimensional rectangle (subscription) $s = ([\ell_1, r_1], [\ell_2, r_2], \ldots, [\ell_\beta, r_\beta])$ is transformed into a 2β-dimensional point $p(s) = (-\ell_1, r_1, -\ell_2, r_2, \ldots, -\ell_\beta, r_\beta)$. The following fact can be easily verified : *for two β-dimensional subscriptions s_1 and s_2, s_1 covers s_2 iff every coordinate of $p(s_1)$ is no less than the corresponding coordinate of $p(s_2)$*. Note that the transformation goes both ways; it is shown [EO82] that 2β-dimensional point dominance can be reduced to the β-dimensional rectangle enclosure (or subscription covering) problem. Henceforth, we consider the following point dominance formulation of subscription covering.

Problem 1 (Point Dominance) *Index a set P of points in d dimensional space to answer the following query efficiently. Given a d dimensional point $x = (x_1, x_2, \ldots, x_d)$, report any point in P that lies in the region $([x_1, \infty], [x_2\infty], \ldots, [x_d, \infty])$. If there are no points in the region, then report "empty".*

Note that we use ∞ to denote the maximum value that can be taken by a coordinate along a dimension. This maximum may be different for different dimensions. Our formulation of approximate subscription covering is through the following relaxed version of point dominance, called *ϵ-approximate point dominance*, for a user specified ϵ.

Problem 2 (ϵ-Approximate Point Dominance) *Index a set P of points in d dimensional space to answer the following query efficiently. Given a user defined parameter $0 < \epsilon < 1$, and a d dimensional query point $x = (x_1, x_2, \ldots, x_d)$, search a subset of the region $([x_1, \infty], [x_2, \infty], \ldots, [x_d, \infty])$ whose volume is at least $(1 - \epsilon)$ of the volume of the entire region. If any point was found in the search, return it, and return "empty" otherwise.*

For example, a 0.05-approximate point dominance query searches 95% of the volume of the region that contains points corresponding to covering subscriptions. The only time when it fails is the case when the query subscription is covered, but all covering subscriptions lie in the remaining

5% of the region that has not been searched. If subscriptions are well distributed over the universe, then an approximate point dominance search can be expected to find most existing covering relations between subscriptions.

For a point dominance query, let b_{max} and b_{min} denote the number of bits required to represent the longest and the shortest sides respectively, of the query rectangle. The aspect ratio α of the query rectangle is defined as $\alpha = b_{max} - b_{min}$[1]. Informally, the aspect ratio is small (close to zero) when the sides of the query region are approximately the same length and large when they are of significantly different lengths.

We consider indexes for approximate point dominance based on the Z space filling curve. The Z curve has been used in a variety of indexing applications, including commercial data products such as Oracle [Ora01, GG98]. Other popularly used SFCs are the Hilbert curve [Hil91] and the Gray code curve [Fal86]. It has been observed [MJFS01] that the performance of the Z and Hilbert curves for many indexing applications are within a constant fraction of each other.

1.2 Our Contributions

We introduce the notion of *approximate covering* to optimize subscription propagation in content-based publish-subscribe systems. Using the point dominance formulation of subscription covering, we show the following. *For point dominance queries where the aspect ratio of the query region is small, approximate point dominance is much cheaper than exhaustive point dominance.* More precisely,

1. The worst case time complexity of an ϵ-approximate point dominance query in d dimensions using the Z SFC is $O\left[\log \frac{d}{\epsilon} \cdot \left(2^{\alpha+1}\frac{d}{\epsilon}\right)^{d-1}\right]$

2. In contrast, the worst case time complexity of an exhaustive point dominance query using the Z SFC is $\Omega\left[\left(2^{\alpha-1}\ell\right)^{d-1}\right]$ where ℓ is the length of the shortest side of the query rectangle.

3. We present a simple algorithm for approximate covering detection based on the Z space filling curve.

[1] In two dimensional space, the aspect ratio of a rectangle is traditionally defined as the ratio of the longer to the shorter side. Our definition of aspect ratio is approximately the logarithm (to base 2) of the traditional definition. This definition leads to a convenient statement of our results.

Somewhat surprisingly, this shows that for a point dominance query with a small aspect ratio, the complexity of an ϵ-approximate query is *independent of the side lengths of the query region*, while the complexity of an exhaustive point dominance query increases as the $(d-1)$th power of the smallest side length of the query region. This implies that an ϵ-approximate query (for a constant ϵ) is much cheaper than an exhaustive query. Further, we can expect that the benefits of approximate covering over exhaustive covering will be more pronounced as the query region gets larger. For query subscriptions with a small aspect ratio, approximate covering can yield most of the benefits of exhaustive covering at a small fraction of the cost, thus making a strong case for using approximate covering in optimizing content-based routing.

If however, the aspect ratio of the query rectangle was large, then the term 2^{α} will dominate the above expressions, and though approximate covering is still cheaper than exhaustive covering, the benefits will not be as much as the case of small aspect ratio. An extreme case in two dimensions is a $M \times 1$ rectangle, which is not efficiently handled by most popular SFCs.

1.3 Related Work

From a worst-case time complexity perspective, the current best solution to point-dominance problem (see [PS85][Ch. 8], [Wil84, WL85]) over a set of n of points in d dimensions has a query time of $O(\log^{d-1} n)$, insertion and deletion times of $O(\log^{d} n)$. A serious limitation of this solution is the space complexity, which is $O(n \log^{d} n)$, making them impractical for use in a pub-sub system. For example, with 10^4 subscriptions each with 4 attributes, the space requirement is easily outside the capacity of the main memory.

Existing solutions to the problem of subscription covering [LHJ05, TE04] do not provide any formal analysis of the performance. In a recent work, Ouksel *et. al.*[OJPA06] consider a relaxed notion of subscription covering and give a probabilistic algorithm for covering detection. The complexity is $O(nm)$, where n is the number of subscriptions and and m is the number of attributes. To our knowledge, ours is the first algorithm for exact or approximate covering with a time complexity that is sublinear in the number of subscriptions being indexed.

Though there have been numerous applications of SFCs for indexing multidimensional data and corresponding experimental analysis, there has been relatively little work on a formal analysis of their performance. Moon *et al.*[MJFS01] present an analysis of the clustering properties of the

Hilbert SFC. They show that given a query region which is a high dimensional rectangle, the average number of clusters of points inside the rectangle is proportional to the surface area of the query rectangle. Their analysis considers exhaustive search while we consider approximate search. Tirthapura *et. al.*[TSA06] show a formal analysis of space filling curves for parallel domain decomposition.

Roadmap. The rest of this paper is organized as follows. In Section 2 we review space filling curves. In Section 3, we first present some intuition as to why approximate point dominance may be cheaper than exhaustive point dominance, and then present the upper bound on the cost of approximate point dominance. This is followed by an analysis of a lower bound on the cost of exhaustive point dominance in Section 4. We then sketch our algorithm for approximate point dominance in Section 5. Section 6 concludes our work.

2 Space Filling Curves

We consider a d-dimensional universe $2^k \times 2^k \cdots \times 2^k$. Note that the number of dimensions d is twice the number of attributes in a subscription. Each element of this universe $p = (x_1, x_2, \ldots, x_d)$, where for all $i = 1 \ldots d$, $x_i \in [0, 2^k - 1]$, is called a *cell*. The SFC imposes a linear order on all 2^{kd} cells. Henceforth we use the term *cube* to refer to a cube in d dimensions and *rectangle* to refer to a rectangle in d dimensions.

Most SFCs used for indexing including the Z curve [Mor66] and the Hilbert curve [Hil91], utilize a recursive partitioning of the universe. The universe is first divided into 2^d cubes, each of side length 2^{k-1}, by bisecting along every dimension. Each resulting cube is recursively divided $k-1$ times until we are left with unit cubes. We use the term *standard cube* to refer to each intermediate cube resulting from this process. When the cube is recursively decomposed $\ell \leq k$ times, there are $2^{d\ell}$ standard cubes each containing $2^{d(k-\ell)}$ cells. Each such cube is referred to as a *standard cube at level ℓ*. Standard cubes at level k are the individual cells.

Lemma 2.1 *Let C and D be two standard cubes which are not equal to each other. Then either C contains D or D contains C or C and D are disjoint from each other.*

Proof: The recursive partitioning of the universe can be visualized as a tree whose root is the

entire universe, and whose leaves are individual cells. Both C and D are nodes in this tree. Since $C \neq D$, the following are the only possibilities (1)C is an ancestor of D, or (2)D is an ancestor of C, or (3)C and D lie in different subtrees. In the first case, C contains D, in the second case D contains C and in the third case C and D are disjoint from each other. ■

Each standard cube at level $\ell \leq k$ is assigned a unique $d\ell$ bit number called its *key*, which defines its position in the total order. Different SFCs differ in the assignment of keys to different standard cubes at the same level. The input points are sorted according to the keys of the cells containing them, and stored in a one-dimensional data structure called the *SFC array* (note that the SFC array could be implemented using any dynamic unidimensional data structure such as a binary tree or a skip list).

A *run* is defined as a set of cells that are consecutively ordered by the SFC. For example, in Figure 1, for the rectangular region, there are three runs in the Z curve, and two runs in the Hilbert curve. All points belonging to a run appear as a contiguous segment in the SFC array. Accessing a run in the SFC requires two binary searches on the keys corresponding to the first and last cells in the run. Hence, an operation on a run, such as examining if a run is empty or not, is very efficient, whether the run is small or large. Hence, the performance of an SFC based query over a region depends on the minimum number of runs the region can be decomposed into. The following fact is true for the Z SFC, and for all SFCs that use a recursive partitioning of the universe. Informally, once an SFC enters a standard cube, it will leave the standard cube only after visiting every cell inside it.

Fact 2.1 *A standard d-cube is a single run.*

3 Upper Bound for Approximate Point Dominance

In this section we derive an upper bound for the cost of an approximate point dominance query.

3.1 Intuition

The performance of the space filling curve on a point dominance query, whether exhaustive or approximate, depends on how many runs the query region can be partitioned into. Accessing

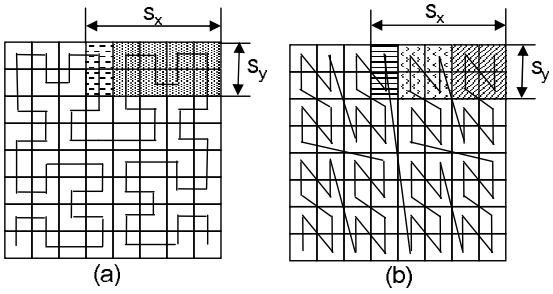

Figure 1: For the same $S_x \times S_y$ rectangle, there are (a)Two runs for the Hilbert SFC and (b)Three runs for the Z SFC

different runs costs the same, but the volume covered by different runs can be vastly different. As a result, the regions that are considered by the approximate and exhaustive point dominance queries may differ only slightly in terms of volume, but widely in terms of the number of runs required to cover them, and hence in the cost of processing.

For example, consider a universe indexed by the Z curve. Figure 2 shows two query regions, each corresponding to a different point dominance query. The first query region is a square of size 256×256, and the second query region is of size 257×257. For the first query, there is a single run that exactly equals the query region, and the cost of answering this query is very small. On the other hand, the number of runs required to exhaustively cover the second query region is 385, since we are forced to cover the periphery of the region using very short runs. In fact, it will be shown (Section 4) that the cost of exhaustively covering a d-dimensional rectangle is proportional to the surface area (the perimeter, in two dimensions) of the rectangle. However, for the second query region, one of the runs covers more than 99% of the query region, while most of the other smaller runs individually cover only 0.015% of the query region. If we only wanted a 0.01-approximate point dominance search, we would be done if we only examined the largest run, and ignored the rest. Thus an approximate point dominance search would be much faster than an exhaustive one for the second query.

The algorithm for approximate point dominance will select a subspace of the query region such that the volume of the subspace is at least $(1 - \epsilon)$ fraction of the volume of the query region, but the number of runs required to cover it is much smaller. We now describe the way in which this subspace is selected.

Given a point $p = (x_1, x_2, \ldots, x_d)$, the exhaustive point dominance asks for any point in the

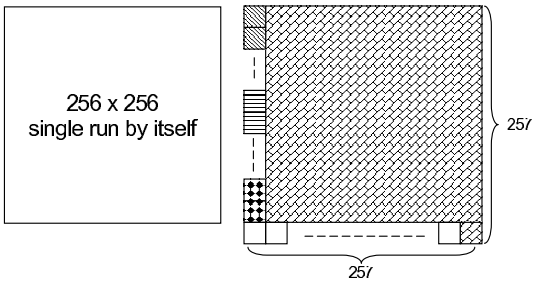

Figure 2: Two example point dominance queries for the Z curve. Standard cubes belonging to the same run are drawn using identical patterns.

rectangle $([x_1, 2^k - 1], [x_2, 2^k - 1], \ldots, [x_d, 2^k - 1])$. We refer to such a rectangle as an *extremal rectangle*, since one of its vertices is the point $(2^k - 1, 2^k - 1, \ldots, 2^k - 1)$. Note that an extremal rectangle can be completely specified through specifying its lengths along each dimension, since one of its vertices is fixed. Let $\ell = (\ell_1, \ell_2, \ldots, \ell_d)$ be a vector where for each $i = 1, \ldots, d$, $1 \leq \ell_i \leq 2^k$. We use $R(\ell)$ to denote an extremal rectangle, whose side lengths along dimensions $1, 2, \ldots, d$ are $\ell_1, \ell_2, \ldots, \ell_d$ respectively.

For a positive integer x, let $b(x)$ denote the number of bits in the binary representation of x, where the most significant bit is 1. For example, $b(9) = 4$. For positive integer x and $m < b(x)$, let $t(x, m)$ denote the integer formed by retaining the m most significant bits in x and setting the rest to zero. When the input is a vector, the operator $t()$ will be applied to each element in the vector. For example, $t(\ell, m) = (t(\ell_1, m), t(\ell_2, m), \cdots, t(\ell_d, m))$.

Given an initial query region $R(\ell)$, the approximate point dominance query considers a smaller extremal rectangle $R(t(\ell, m))$ that is completely contained within $R(\ell)$. For ease of notation, we use $R^m(\ell)$ to represent $R(t(\ell, m))$. The parameter m is chosen as a function of ϵ (the user desired coverage) such that the chosen rectangle covers at least $(1 - \epsilon)$ fraction of the volume of $R(\ell)$.

3.2 Upper Bound

We now formally prove an upper bound on the cost of approximate point dominance for the Z SFC. The main result in this Section is Theorem 3.1.

Definition 3.1 *For rectangle T, $cubes(T)$ is defined as the minimum number of standard cubes into which T can be partitioned, and $runs(T)$ is defined as the minimum number of runs that make*

up all cells of T in the SFC array.

Lemma 3.1 *For any rectangle T, $cubes(T) \geq runs(T)$.*

Proof: Consider a partitioning of T into $cubes(T)$ standard cubes. From Fact 2.1 each standard cube in such a partitioning corresponds to a single run. This immediately leads to a clustering of all cells that make up T in the SFC array into $cubes(T)$ runs. Thus it must be true that $runs(T) \leq cubes(T)$. ■

The worst-case cost of approximate point dominance is equal to $runs(R^m(\ell))$. According to Lemma 3.1, this is bounded by $cubes(R^m(\ell))$. So our strategy for the proof goes as follows. Lemma 3.3 shows that a greedy algorithm can partition an arbitrary region into a minimum number of standard cubes. Lemma 3.4 classifies the obtained standard cubes so that we are able to totalize the amount in Lemma 3.7. Note that the proofs are very general so that they remain valid for other SFC, such as Hilbert curve, which is based on the recursive partitioning of the space.

As the first step, we decide what value of m produces the required space coverage. The following lemma shows that if we truncated each side length of $R(\ell)$ down to its $\log_2 \frac{2d}{\epsilon}$ most significant bits, then the volume of the resulting rectangle is within $(1-\epsilon)$ of the volume of $R(\ell)$.

Lemma 3.2 *Let $0 < \epsilon < 1$. If $m \geq \log_2 \frac{2d}{\epsilon}$, then $\frac{vol(R^m(\ell))}{vol(R(\ell))} \geq 1 - \epsilon$*

Proof: Let $\gamma = \frac{vol(R^m(\ell))}{vol(R(\ell))}$.

$$\gamma = \prod_{i=1}^{i=d} \frac{t(\ell_i, m)}{\ell_i}$$

$$\frac{t(\ell_i, m)}{\ell_i} \geq \frac{t(\ell_i, m)}{t(\ell_i, m) + 2^{b(\ell_i)-m}} = \frac{1}{1 + \frac{2^{b(\ell_i)-m}}{t(\ell_i, m)}} \geq \frac{1}{1 + \frac{2^{b(\ell_i)-m}}{2^{b(\ell_i)-1}}} = \frac{1}{1 + \frac{1}{2^{m-1}}}$$

Since $1 + x \leq e^x$ for all real x,

$$\frac{t(\ell_i, m)}{\ell_i} \geq e^{\frac{-1}{2^{m-1}}} = e^{\frac{-2}{2^m}}$$

Thus,

$$\gamma \geq \prod_{i=1}^{i=d} e^{\frac{-2}{2^m}} = e^{\frac{-2d}{2^m}} \geq 1 - \frac{2d}{2^m}$$

Since $m \geq \log_2 \frac{2d}{\epsilon}$, we have $\frac{2d}{2^m} \leq \epsilon$, and thus $\gamma \geq 1 - \epsilon$ ■

Next we consider the partitioning of a given extremal rectangle into standard cubes. It is always trivially possible to partition any rectangle into a set of standard cubes by breaking it up into individual cells (note that each cell is a standard cube). We now describe a greedy algorithm that outputs the partition of any region (which is not necessarily a rectangle) into a *minimum* number of standard cubes. Suppose it is required to partition a region R into standard cubes. The greedy algorithm works as follows. *If R is empty, then the algorithm outputs nothing, and exits. Otherwise, the largest standard cube that fits within R, say C, is chosen and output. The algorithm is then recursively applied on input $R - C$.*

Lemma 3.3 *The greedy algorithm, when applied to region R, produces an optimal partition of R into a minimum number of standard cubes.*

Proof: Proof by induction on $vol(R)$, the number of cells in R. For the base case, where $vol(R) = 0$, the algorithm is obviously correct. For the inductive case, assume that for every region R such that $vol(R) < \kappa$, the greedy algorithm decomposes R into the minimum number of standard cubes.

Consider a region R such that $vol(R) = \kappa$. Let C be the largest standard cube completely contained within R. Let D denote a partition of R into the minimum number of standard cubes. We use proof by contradiction to show that E must contain C as an element. Suppose $C \notin E$. Let $E' \subset E$ represent the set of standard cubes within E that contain cells that makeup C. Clearly E' is non-empty. Since each element of E' intersects C, and none of them can equal C, it must be true from Lemma 2.1 that every element of E' is fully contained within C. This yields a decomposition of R into fewer number of standard cubes than E by replacing E' with a single cube C, contradicting the optimality of E. Thus E must contain C, and the first step of the greedy algorithm is proved correct. The remaining region $(R - C)$ has fewer than κ cells, since C is non-empty. By the inductive hypothesis, the greedy algorithm generates an optimal decomposition of $(R - C)$, and the proof is complete. ■

Let D represent the set of standard cubes resulting from a decomposition of $R(\ell)$ into standard cubes based on the greedy algorithm. Let D_i represent the subset of D consisting of standard cubes of side length 2^i. For integer x, let x_j denote the jth bit in the binary representation of x. For integer x, let $S_i(x) = \sum_{j=i}^{j=b(x)-1} x_j 2^j$, i.e. $S_i(x)$ is the result of choosing only the most significant bits in the binary representation of x starting from x_i onwards. When the input is a vector, the

operator $S_i()$ will be applied to each element in the vector. Thus $S_i(\ell) = (S_i(\ell_1), S_i(\ell_2), \cdots, S_i(\ell_d))$. Let $\ell_{i,j} = \ell_{ij}$, i.e. the jth bit of ℓ_i. For $j = 1, \ldots, d$, indicator variable O_j is defined as follows. $O_j = 0$ if $\ell_{i_j} = 0$ for all $i = 1, \ldots, d$; $O_j = 1$ if $\ell_{i,j} = 1$ for some i, $1 \leq i \leq d$. Assume w.l.o.g. $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_d$. The following lemma gives a precise characterization of the type and location of the standard cubes resulting from an optimal partition of $R(\ell)$.

Lemma 3.4 *For $b(\ell_1) \leq i \leq b(\ell_d) - 1$, D_i is empty. For $0 \leq i \leq b(\ell_1) - 1$*

1. *D_i is non-empty if and only if $O_i = 1$.*
2. *The region occupied by $\cup_{j=i}^{j=b(\ell_1)-1} D_j$ is the extremal rectangle $R(S_i(\ell))$.*

Proof: First we prove by contradiction that D_i is empty for $b(\ell_1) \leq i \leq b(\ell_d) - 1$. Suppose D_j is non-empty for some $j, b(\ell_1) \leq j \leq b(\ell_d) - 1$. Let $C_j \in D_j$ be a standard cube of side length 2^j. The projection of C_j on the first dimension dimension is a line segment of length $2^j \geq 2^{b(\ell_1)}$. But ℓ_1 is at most $2^{b(\ell_1)} - 1$ since it is a $b(\ell_1)$ bit number, leading to a contradiction. Thus D_i must be empty for $b(\ell_1) \leq i \leq b(\ell_d) - 1$.

For the rest of i, $b(\ell_1) - 1 \geq i \geq 0$, we use proof by reverse induction on i, starting from $i = b(\ell_1) - 1$ and going down to $i = 0$.

We first consider the base case: $i = b(\ell_1) - 1$. We have $O_{b(\ell_1)-1} = 1$, because $\ell_{1,b(\ell_1)-1} = 1$. Since $S_{b(\ell_1)-1}(\ell_j)$ is non-zero and no more than ℓ_j for $1 \leq j \leq d$, it must be true that the extremal rectangle $R(S_{b(\ell_1)-1}(\ell))$ is not empty and is fully contained inside $R(\ell)$. Moreover, $S_{b(\ell_1)-1}(\ell_j)$ is divisible by $2^{b(\ell_1)-1}$ for $1 \leq j \leq d$. So $R(S_{b(\ell_1)-1}(\ell))$ can be decomposed into the union of standard cubes of side length $2^{b(\ell_1)-1}$. We know from the above that D_i is empty for $b(\ell_d) - 1 \geq i \geq b(\ell_1)$. This indicates that the largest standard cube which can fit into $R(\ell)$ has a side length of $2^{b(\ell_1)-1}$. Such standard cubes will be chosen by the greedy algorithm if they exist. Since $R(S_{b(\ell_1)-1}(\ell))$ is not empty, we know $D_{b(\ell_1)-1}$ is also non-empty.

Finally we claim there is no standard cube of side length $2^{b(\ell_1)-1}$ in the region $R(\ell) - R(S_{b(\ell_1)-1}(\ell))$. We use proof by contradiction. Suppose the specified region contains one standard cube $C \in D_{b(\ell_1)-1}$. Since two standard cubes cannot intersect each other (Lemma 2.1), there can be no intersection between C and $R(S_{b(\ell_1)-1}(\ell))$. Under this condition, there must exist at least one dimension onto which the projections of C and $R(S_{b(\ell_1)-1}(\ell))$ are disjoint. Proof by contradiction. Suppose the projections of C and $R(S_{b(\ell_1)-1}(\ell))$ overlap on every dimension. Let $[l_i, r_i]$

denote the overlapping segment between the two projections along the ith dimension. As a result, $[l_1, r_1] \times [l_2, r_2] \times \cdots \times [l_d, r_d]$ forms a rectangle which is shared by both rectangles, which contradicts the fact that they don't intersect. Therefore the projections of C and $R(S_{b(\ell_1)-1}(\ell))$ must be disjoint along some dimension.

Assume w.l.o.g. that the projections of C and $R(S_{b(\ell_1)-1}(\ell))$ are disjoint along dimension 1. The combined lengths of their projections along dimension 1 is $S_{b(\ell_1)-1}(\ell_1) + 2^{b(\ell_1)-1} = 2^{b(\ell_1)}$. However ℓ_1 can be no more than $2^{b(\ell_1)} - 1$. We reached a contradiction. Therefore the region $R(\ell) - R(S_{b(\ell_1)-1}(\ell))$ can not contain C. As a consequence, the region occupied by $D_{b(\ell_1)-1}$ corresponds to the extremal rectangle $R(S_{b(\ell_1)-1}(\ell))$. This proves the base case.

For the inductive case, assume that the theorem remains true for every i, $b(\ell_1) - 1 \geq i \geq \kappa + 1$. We now consider the case $i = \kappa$. We study the following two cases.

Case I: $O_\kappa = 0$. We prove by contradiction that D_κ must be empty. Suppose D_κ is not empty and $C_\kappa \in D_\kappa$ be a standard cube of side length 2^κ. The inductive hypothesis tells us $\cup_{j=\kappa+1}^{j=b(\ell_1)-1} D_j$ forms an extremal rectangle $R(S_{\kappa+1}(\ell))$. We are able to show, by following the analysis for the base case, the remaining region $R(\ell) - R(S_{\kappa+1}(\ell))$ can not contain C_κ. This means D_κ must be empty. Since D_κ is empty, the inductive hypothesis can be directly extended to show that standard cubes in D with a side length of 2^κ or higher form an extremal rectangle $R(S_\kappa(\ell))$.

Case II: $O_\kappa = 1$. We now consider the case $O_\kappa = 1$. Consider the extremal rectangles $R(S_\kappa(\ell))$ and $R(S_{\kappa+1}(\ell))$. Since $S_{\kappa+1}(\ell_i) \leq S_\kappa(\ell_i) \leq \ell_i$, it must be true that $R(S_{\kappa+1}(\ell))$ is fully contained in $R(S_\kappa(\ell))$, which is in turn fully contained within $R(\ell)$. Since $S_\kappa(\ell_i)$ is divisible by 2^κ for $1 \leq i \leq d$, so $R(S_\kappa(\ell))$ can be decomposed into a union of standard cubes of side length 2^κ. Similarly, it can also be shown that $R(S_{\kappa+1}(\ell))$ can be decomposed into a union of standard cubes of side length 2^κ. Since $R(S_{\kappa+1}(\ell))$ is fully contained within $R(S_\kappa(\ell))$, the set of standard cubes in $R(S_{\kappa+1}(\ell))$ is a subset of the standard cubes in $R(S_\kappa(\ell))$. Thus the region $R(S_\kappa(\ell))$ - $R(S_{\kappa+1}(\ell))$ can also be decomposed into multiple standard cubes of side length 2^κ. Furthermore because of $O_\kappa = 1$, we must have $\ell_{i,\kappa} = 1$ for some i and in turn $S_\kappa(\ell_i) - S_{\kappa+1}(\ell_i) > 0$. So the region $R(S_\kappa(\ell))$ - $R(S_{\kappa+1}(\ell))$ is not empty. We can follow an analysis that is similar to the base case to show that the remaining region $R(\ell) - R(S_\kappa(\ell))$ does not contain any standard cube of side length 2^κ. This completes the proof that D_κ is non-empty and the union of standard cubes of side length 2^κ or higher in D form the extremal rectangle $R(S_\kappa(\ell))$. ■

For every $i = 0, \ldots, b(\ell_1) - 1$, let $N_i = |D_i|$, i.e. the number of standard cubes of side length 2^i in the decomposition of $R(\ell)$ into a minimum number of standard cubes.

Lemma 3.5 *For every $i = 0, \ldots, b(\ell_1) - 1$, we have the following. If $O_i = 0$, then $N_i = 0$. If $O_i = 1$, then*

$$N_i = \frac{\prod_{j=1}^{j=d} S_i(\ell_j) - \prod_{j=1}^{j=d} S_{i+1}(\ell_j)}{2^{id}}$$

Proof: From Lemma 3.4, we know that if $O_i = 0$, then $N_i = 0$. If $O_i = 1$, all standard cubes in D_i lie in the region $R(S_i(\ell)) - R(S_{i+1}(\ell))$. The expression for N_i can be derived through dividing the difference in the volumes of the two rectangles by the volume of a standard cube of side length 2^i. ■

Lemma 3.6 *$cubes(R^m(\ell))$ is maximized iff (1) $\ell_{j,x} = 1$ for $x \in [b(\ell_j) - m, b(\ell_j) - 1]$ and $j \in [1, d]$ and (2) $b(\ell_j) = b(\ell_d)$ for $1 < j < d$.*

Proof: From Lemma 3.4, we have $cubes(R^k(\ell)) = \sum_{i=b(\ell_1)-k}^{i=b(\ell_1)-1} N_i$. Using Lemma 3.5,

$$\begin{aligned}
N_i &= \frac{1}{(2^i)^d} \left(\prod_{j=1}^{j=d} S_i(\ell_j) - \prod_{j=1}^{j=d} S_{i+1}(\ell_j) \right) \\
&= \frac{1}{(2^i)^d} \left[\prod_{j=1}^{j=d} (S_{i+1}(\ell_j) + \ell_{j,i}) - \prod_{j=1}^{j=d} S_{i+1}(\ell_j) \right] \\
&= \frac{1}{(2^i)^d} \left[\prod_{\substack{j=1 \\ j \neq 1}}^{j=d} S_{i+1}(\ell_j) \cdot \ell_{1,i} + \cdots + \prod_{\substack{j=1 \\ j \neq 1,2}}^{j=d} S_{i+1}(\ell_j) \cdot \prod_{j=1}^{j=2} \ell_{j,i} + \cdots + S_{i+1}(\ell_1) \cdot \prod_{j=2}^{j=d} \ell_{j,i} + \cdots + 1 \right]
\end{aligned}$$

It can be verified that each individual component in the above expression is maximized when $(1)\ell_{j,x} = 1$ for $x \in [i, b(\ell_j) - 1]$ and $j \in [1, d]$ $(2)b(\ell_j) = b(\ell_d)$ for $1 < j < d$. Thus, N_i is maximized under the same conditions. Since this is true for all i ranging from $b(\ell_1) - k$ till $b(\ell_1) - 1$, the lemma follows. ■

Recall that $\alpha = b(\ell_d) - b(\ell_1)$ is the aspect ratio of the rectangle, and $0 < \epsilon < 1$ is a user parameter indicating the desired coverage of the approximate query.

Lemma 3.7 *$cubes(R^m(\ell)) < m \cdot [2^\alpha (2^m - 1)]^{d-1}$*

Proof: We consider two cases: (1)$k < \alpha$ and (2)$k \geq \alpha$. To find an upper bound on $cubes(R^k(\ell))$, we assume vector ℓ satisfies the conditions specified by Lemma 3.6.

Case 1: $k < \alpha$.

According to Lemma 3.5, we have for $i \in [b(\ell_1) - k, b(\ell_1) - 1]$,

$$N_i = \prod_{j=2}^{j=d} \frac{S_i(\ell_j)}{2^i} \cdot \frac{S_i(\ell_1)}{2^i} - \prod_{j=2}^{j=d} \frac{S_{i+1}(\ell_j)}{2^i} \cdot \frac{S_{i+1}(\ell_1)}{2^i}$$

According to Lemma 3.6, we have $\ell_{j,x} = 1$ for $x \in [b(\ell_d) - k, b(\ell_d) - 1]$ and $1 < j \leq d$.

Since $k < \alpha$, it must be true that $b(\ell_d) - k > b(\ell_1) > i$. This leads to $S_i(\ell_j) = S_{b(\ell_d)-k}(\ell_d)$ for $1 < j \leq d$.

$$\begin{aligned} N_i &= \left(\frac{S_{b(\ell_d)-k}(\ell_d)}{2^i}\right)^{d-1} \cdot \left(\frac{S_i(\ell_1)}{2^i} - \frac{S_{i+1}(\ell_1)}{2^i}\right) = \left(\sum_{x=b(\ell_d)-k}^{x=b(\ell_d)-1} 2^{x-i}\right)^{d-1} \\ &= \left(2^{b(\ell_d)-i} - 2^{b(\ell_d)-k-i}\right)^{d-1} = \left[2^{b(\ell_d)-i} \cdot \left(1 - \frac{1}{2^k}\right)\right]^{d-1} \\ &< \left[2^{\alpha+k} \cdot \left(1 - \frac{1}{2^k}\right)\right]^{d-1} < \left[2^{\alpha}(2^k - 1)\right]^{d-1} \end{aligned}$$

$$cubes(R^k(\ell)) = \sum_{i=b(\ell_1)-k}^{i=b(\ell_1)-1} N_i < \sum_{i=b(\ell_1)-k}^{i=b(\ell_1)-1} \left[2^{\alpha}(2^k - 1)\right]^{d-1} = k \cdot \left[2^{\alpha}(2^k - 1)\right]^{d-1}$$

Case 2: $k \geq \alpha$.

According to Lemma 3.5, we have for $i \in [b(\ell_1) - k, b(\ell_1) - 1]$,

$$N_i = \prod_{j=2}^{j=d} \frac{S_i(\ell_j)}{2^i} \cdot \frac{S_i(\ell_1)}{2^i} - \prod_{j=2}^{j=d} \frac{S_{i+1}(\ell_j)}{2^i} \cdot \frac{S_{i+1}(\ell_1)}{2^i}$$

We further split the derivation into two subcases.

Case 2.1 $i \in [b(\ell_d) - k, b(\ell_1) - 1]$.

According to Lemma 3.6, we have $\ell_{j,x} = 1$ for $x \in [b(\ell_d) - k, b(\ell_d) - 1]$ and $1 < j \leq d$.

Since $i \geq b(\ell_d) - k$, we further get $\ell_{j,x} = 1$ for $x \in [i, b(\ell_d) - 1]$ and $1 < j \leq d$. This leads to $S_i(\ell_j) = S_i(\ell_d)$ for $1 < j \leq d$.

$$N_i = \left(\frac{S_i(\ell_d)}{2^i}\right)^{d-1} \cdot \frac{S_i(\ell_1)}{2^i} - \left(\frac{S_{i+1}(\ell_d)}{2^i}\right)^{d-1} \cdot \frac{S_{i+1}(\ell_1)}{2^i}$$

Note that

$$\frac{S_i(\ell_d)}{2^i} = \frac{S_{i+1}(\ell_d)}{2^i} + 1 \ , \frac{S_i(\ell_1)}{2^i} = \frac{S_{i+1}(\ell_1)}{2^i} + 1$$

We define $X = \frac{S_i(\ell_d)}{2^i}$ and $Y = \frac{S_i(\ell_1)}{2^i}$.

$$X = \frac{S_i(\ell_d)}{2^i} = \sum_{x=i}^{x=b(\ell_d)-1} 2^{x-i} = 2^{b(\ell_d)-i} - 1$$

Similarly $Y = 2^{b(\ell_1)-i} - 1$. We have $X \approx 2^\alpha \cdot Y$.

$$\begin{aligned}
N_i &= X^{d-1} \cdot Y - (X-1)^{d-1} \cdot (Y-1) \\
&= (X^{d-1} - (X-1)^{d-1}) \cdot Y + (X-1)^{d-1} \\
&= [X^{d-2} + X^{d-3} \cdot (X-1) + \cdots + (X-1)^{d-2}] \cdot Y + (X-1)^{d-1} \\
&= X^{d-2} \cdot \left[1 + \frac{X-1}{X} + \cdots + \left(\frac{X-1}{X}\right)^{d-2}\right] \cdot Y + (X-1)^{d-1} \\
&< X^{d-2} \cdot (d-1) \cdot Y + (X-1)^{d-1} \\
&< X^{d-2} \cdot [Y(d-1) + X]
\end{aligned}$$

If we assume $2^\alpha > (d-1)$, we have $[Y(d-1) + X] < 2X$.

$$N_i < 2 \cdot X^{d-1} = 2 \cdot (2^{b(\ell_d)-i} - 1)^{d-1} < 2 \cdot (2^k - 1)^{d-1}$$

Case 2.2: $i \in [b(\ell_1) - k, b(\ell_d) - k)$.

The derivation is similar to Case 1. We have $N_i < \left[2^\alpha(2^k - 1)\right]^{d-1}$

By combining case 2.1 and case 2.2, we have

$$\begin{aligned}
cubes(R^k(\ell)) &< \sum_{i=b(\ell_d)-k}^{i=b(\ell_1)-1} 2 \cdot (2^k - 1)^{d-1} + \sum_{i=b(\ell_1)-k}^{i=b(\ell_d)-k-1} [2^\alpha(2^k - 1)]^{d-1} \\
&= 2(k - \alpha) \cdot (2^k - 1)^{d-1} + \alpha \cdot [2^\alpha(2^k - 1)]^{d-1} \\
&< (\alpha \cdot 2^{\alpha(d-1)} + 2k) \cdot (2^k - 1)^{d-1} \\
&< (k \cdot 2^{\alpha(d-1)} + 2k) \cdot (2^k - 1)^{d-1} \\
&= k \cdot (2^{\alpha(d-1)} + 2) \cdot (2^k - 1)^{d-1} \\
&\approx k \cdot \left[2^\alpha(2^k - 1)\right]^{d-1}
\end{aligned}$$

■

Theorem 3.1 *For any SFC that is based on a recursive partitioning of the universe (such as the Z curve and the Hilbert curve), the cost of an ϵ-approximate point dominance query is* $O(\log \frac{d}{\epsilon} \cdot \left(2^{\alpha+1}\frac{d}{\epsilon}\right)^{d-1})$

Proof: Using Lemma 3.7 and since $runs(R^m(\ell)) \leq cubes(R^m(\ell))$ (Lemma 3.1), we have $runs(R^m(\ell)) < m \cdot [2^\alpha(2^m-1)]^{d-1}$. From Lemma 3.2, if we choose $m = \log_2 \frac{2d}{\epsilon}$ we are guaranteed that $R^m(\ell)$ covers at least $(1-\epsilon)$ fraction of the volume of $R(\ell)$. Thus the number of runs that have to be accessed for an ϵ-approximate point-dominance query is no more than $\log_2 \frac{2d}{\epsilon} \cdot \left[2^\alpha\left(\frac{2d}{\epsilon}-1\right)\right]^{d-1}$, which yields the desired upper bound.

■

4 Lower Bound for Exhaustive Point Dominance

In this section, we prove a lower bound on the cost of exhaustive point dominance for the Z curve (Theorem 4.1). The worst case cost is equal to $runs(R(\ell))$. However, estimating the size of $runs(R(\ell))$ is much harder than estimating the size of $cubes(R(\ell))$, there is no simple characterization for runs like the "greedy" characterization for cubes. For proving a lower bound, our strategy is as follows. We use Lemma 3.4 to characterize the standard cubes resulting from a greedy (optimal) partition of $R(\ell)$. Then we show that for a (large) subset of the standard cubes in the partition, no two standard cubes in the subset can belong to the same run in the Z curve. Thus $runs(R(\ell))$ is no less than the size of this subset.

We consider a class of extremal rectangle $R(\ell)$ whose aspect ratio is α. First some integral $0 < \gamma \leq k - \alpha$ is chosen. The smallest side of the rectangle is along dimension d, and its length is $2^\gamma - 1$, i.e. a bit string of γ ones. The length of a side along dimension $i, 1 \leq i < d$ satisfies the following condition: $b(\ell_i) = \gamma + \alpha$ for $1 \leq i < d$. The following notation D_i and O_i are borrowed from Lemma 3.4. Since $\ell_{d,0} = 1$, we have $O_0 = 1$ and D_0 is non empty according to Lemma 3.4. We can partition the region occupied by D_0 into disjoint rectangles which have the following properties. (1)The length of each side of the rectangle is a power of 2. (2)There exists at least one dimension along which the side length of the rectangle is exactly 2^0.

We consider one rectangle $R_0 \subset R(\ell)$, described as follows. For every dimension $1 \leq i < d$, the projection of R_0 along dimension i is the segment $[2^k - 1 - 2^{b(\ell_1)-1}, 2^k - 1]$. The projection of R_0

dimension	coordinates
1	$\overbrace{1\cdots 1\underbrace{*\cdots *}_{b(\ell_1)-1}}^{k}$
$2,\cdots,(d-1)$	same as dimension 1
d	$\overbrace{1\cdots 1\underbrace{0\cdots 01}_{b(\ell_d)}}^{k}$

Figure 3: Coordinates of a cell in R_0. Symbol $*$ can be either 1 or 0. The size of the universe along each dimension is 2^k.

$$\overbrace{\underbrace{1\cdots 1}_{k-b(\ell_1)+1}\cdots\overbrace{1\cdots 1}^{d}}^{d}\overbrace{\underbrace{\times\cdots\times 1}_{\alpha-1}\cdots\overbrace{\times\cdots\times 1}^{d}}^{d}\overbrace{\underbrace{\times\cdots\times 0}_{b(\ell_d)-1}\cdots\overbrace{\times\cdots\times 0}^{d}}^{d}\overbrace{\times\cdots\times 1}^{d}$$

Figure 4: Key of a cell in R_0

along dimension d is the segment $[2^k-\ell_1-1, 2^k-\ell_1]$. Thus, the side length of R_0 along dimension d is 1, and the side length of R_0 along all other dimensions is $2^{b(\ell_1)-1}$.

We now show that no two standard cells in R_0 may belong to the same run in the Z SFC. To see this, consider the keys of the standard cubes within R_0. Since the universe is modeled as a d-dimensional space $2^k \times 2^k \cdots \times 2^k$, the coordinate of any cell along each dimension is a k bit string. Due to our choice of rectangle R_0, the coordinates of each cell in R_0 has the pattern shown in Figure 5(f).

In the Z curve, the key of a cell is formed by interleaving the bits representing its coordinates, starting from dimension 1, and then proceeding to higher dimensions. For example, suppose a cell has coordinates $(3,5) = (011, 101)_2$. Then the corresponding key is $(011011)_2 = 27$. Converting the coordinates in Figure 5(f) to keys, we find that the key of a cell in R_0 must obey the pattern shown in Figure 4.

Lemma 4.1 *No two standard cubes in R_0 belong to the same run on the Z space filling curve.*

Proof: Every standard cube in rectangle R_0 must be a cell, since one side of R_0 has unit length. Let S denote the keys of all cells in R_0. We show that no pair of keys in S belong to the same run in the Z SFC. Consider two keys $s_1, s_2 \in S$. Assume w.l.o.g. $s_1 > s_2$. Since both s_1 and s_2 are odd

numbers (from Figure 4), the difference between them is at least 2, implying that they cannot be adjacent in the SFC.

Consider the cell c corresponding to the key $s_1 - 1$. Cell c must be outside the query rectangle $R(\ell)$, for the following reason. The key of $s_1 - 1$ exhibits the same pattern as shown in Figure 4, except that the least significant bit is 0. By unwinding this key to retrieve the coordinates of the cell, we find that the dth dimension coordinate of c must be a k bit string consisting of $k - b(\ell_d)$ ones followed by $b(\ell_d)$ zeros. The projection of rectangle $R(\ell)$ along dimension d is a line segment of length ℓ_d whose right endpoint is $2^k - 1$ (a bit string of k ones) and left endpoint is $2^k - 1 - \ell_d$ (a bit string of $k - b(\ell_d)$ ones followed by $b(\ell_d) - 1$ zeroes followed by a one). Clearly, the dth dimension coordinate of c cannot lie in this range, and hence c cannot belong to $R(\ell)$.

We have shown that in the linear ordering produced by the Z SFC, between any two standard cubes in R_0, there must be a cell which does not belong to $R(\ell)$. Since we are interested in partitioning only the cells of $R(\ell)$ into runs, without including any cell outside $R(\ell)$, no two standard cubes in R_0 can belong to the same run for the Z SFC. ■

Theorem 4.1 *For any integer $0 \leq \alpha < k$, there exists an extremal rectangle $R(\ell)$ whose aspect ratio is α and the cost of an exhaustive search of $R(\ell)$ using the Z space filling curve is $\Omega\left[\left(2^{\alpha-1} \cdot \ell_d\right)^{d-1}\right]$*

Proof: The cost of an exhaustive search is lower bounded by $runs(R_0)$, since all cells in R_0 have to be examined. From Lemma 4.1 we know $runs(R_0) = cubes(R_0)$. Since each standard cube in R_0 is a cell, $cubes(R_0) = vol(R_0)$

$$vol(R_0) = \prod_{j=1}^{j=d-1} 2^{b(\ell_j)-1} = \left(\frac{2^{b(\ell_d)} \cdot 2^{\alpha}}{2}\right)^{d-1} = \left(\frac{2^{\alpha} \cdot \ell_d}{2}\right)^{d-1}$$

■

5 Algorithm for Approximate Point Dominance

In this section, we sketch a simple algorithm for approximate point dominance based on the Z SFC. The only data structure to maintain is the SFC array, which sorts the points according to their orders on the Z curve. It should be fairly easy to maintain this sorted order, while allowing frequent additions and deletions of points, by using a dynamic ordered data structure such as a balanced binary tree.

Given a point dominance query, our algorithm follows the greedy approach to partition the query region into a minimum number of standard cubes. The algorithm then searches these cubes in the SFC array for covering subscriptions, in the descending order of their volume. Meanwhile it keeps track of the ratio of the volume searched to the volume of the query region. This ratio is updated each time a standard cube is visited. The search terminates either a covering subscription is found or this ratio exceeds $1 - \epsilon$.

The major operation in the above algorithm is to identify standard cubes produced by the greedy decomposition. The goal here is to compute the cubes' keys(Section 2), which are required by the search in the SFC array.

The Z Curve generates the key of a standard cube by treating the coordinates of a cube as binary numbers and interleaving the bits. For example, consider the standard squares in Figure 5(c), which result from 3 rounds of recursive partitioning of the space. We can use 3 bits to denote a square's coordinate on a single dimension. In this picture, square "a" has coordinates $(010, 011)$. Its key with respect to the Z curve is equal to $(001101)_2 = 13$.

Lemma 3.4 in Section 3 states that the standard cubes generated by the greedy algorithm can be classified into different D_i's. Therefore it is sufficient for us to demonstrate how to compute the keys of standard cubes within a particular D_i. We use the following two-phase algorithm. In the first stage, we decompose the space occupied by D_i into disjoint rectangles with the following properties: (1)The length of each side of the rectangle must be a multiple of 2^i so that the entire rectangle is a union of standard cubes in D_i. (2)There exists at least one dimension along which the side length of the rectangle is exactly 2^i. In the second stage, we identify the standard cubes within each rectangle and compute their keys.

Before explaining the details, let us review the definition of some variables: $R(\ell)$ represents the query region, where ℓ is a vector specifying the lengths of the region along each dimension. We assume w.l.o.g. $\ell_1 \leq \ell_2 \ldots \leq \ell_d$. $\ell_{x,y}$ refers to the yth bit w.r.t. the least significant in ℓ_x. $b(\ell_x)$ denotes the number of bits in ℓ_x. d is the dimension of the universe. k indicates the deepest level in a recursive partitioning of the universe induced by the Z curve. Let i be the subscript of D_i.

Based on the constraints, to decide a rectangle's length on dimension x, we only need to consider non-zero bits in ℓ_x, whose index is no less than i. If we create a vector P of size d, such that the xth element in P denotes the index of a non-zero bit we selected from ℓ_x and $P_x \geq i$ for every x,

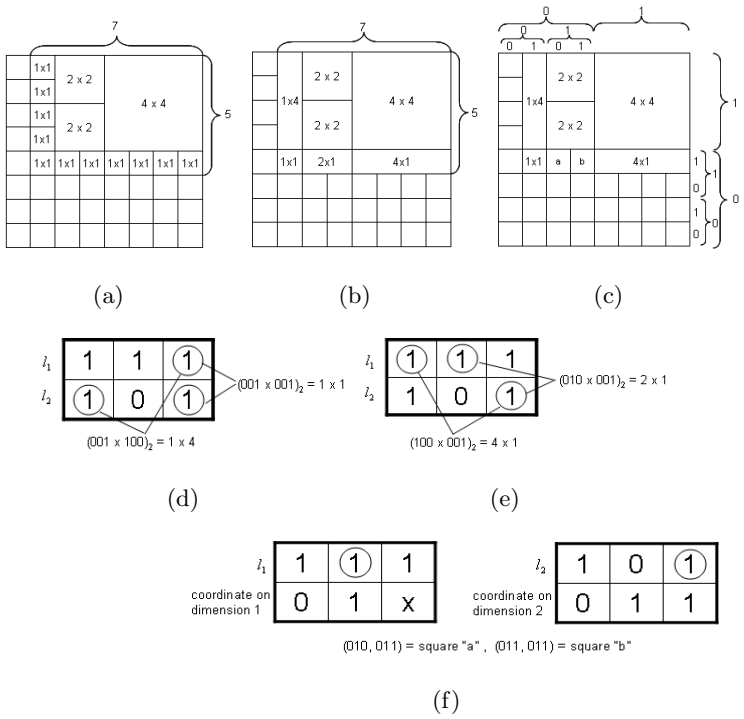

Figure 5: Compute the keys of standard cubes resulted from a greedy decomposition. Symbol $\times$ can be either 1 or 0

then it can be verified that each unique instance of P corresponds to a rectangle lying in the space occupied by D_i.

For example, all the 1×1 squares in Figure 5(a) constitute the space occupied by D_0. We can divide this "L"-shape strip into four rectangles as shown in Figure 5(b). Figure 5(d) and 5(e) illustrate the corresponding instances of P.

Our next step is to compute the keys of standard cubes contained inside a rectangle. Note that the input rectangle is represented by vector P. We create another vector Q to store the coordinates of a cube. It follows that Q_x is a cube's coordinate on dimension x.

We know that standard cubes in D_i result from $(k - i)$ rounds of recursive partitioning of the universe. Thus we can use $(k - i)$ bits to represent a cube's coordinate on a single dimension. In practice, we allocate a string of k bits to store one coordinate. For a standard cube in D_i, only the

highest $(k-i)$ bits are used. We can compute Q based on P and ℓ in the following way:

$$\begin{aligned} Q_{x,y} &= !\ell_{x,y} \text{ (! means negate)}, && \text{for } y \in (P_x, k-1] \\ Q_{x,y} &= \ell_{x,y}, && \text{for } y = P_x \\ Q_{x,y} &= \text{either 0 or 1}, && \text{for } y \in [i, P_x) \end{aligned} \tag{1}$$

For example, the rectangle "2×1" in Figure 5(b) consists of two standard squares "a" and "b" as shown in Figure 5(c). We get the rectangle by selecting the first bit from ℓ_1 and the rightmost bit from ℓ_2. Figure 5(f) shows how Equation 1 can be applied to compute the coordinates of "a" and "b". Once the coordinates of a cube are available, we can get its key by interleaving the bits.

This concludes our algorithm description. The pseudo code is presented in Appendix A.

6 Conclusion

The detection of covering among content-based subscriptions is a hard combinatorial problem. Existing researches concerning this problem are interested in exact answers. However we point out that subscription covering is just an optimization, which doesn't need to be strictly followed all the time. Based on this observation, we introduce the notion of approximate covering to retain most benefits of exact covering at a fraction of its cost.

Our approximate solution is based on Z space filling curve and supports numeric subscriptions. Unlike relevant researches which are limited to empirical study, we formally prove the algorithm's complexity. The analysis shows that when the aspect ratio of the query region is small, approximate covering is much cheaper than exact covering. Furthermore we present a simple implementation of the algorithm so that our results can directly benefit the current implementations of content-based publish/subscribe system.

References

[Cha90a] B. Chazelle. Lower bounds for orthogonal range searching:ii.the arithmetic model. *Journal of the ACM*, 37:439–463, Jul 1990.

[Cha90b] B. Chazelle. Lower bounds for orthogonal range searching:i.the reporting case. *Journal of the ACM*, 37:200–212, Apr 1990.

[CNF01] G. Cugola, E. Di Nitto, and A. Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *IEEE Trans. Softw. Eng.*, 27(9):827–850, 2001.

[CRW04] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *Proceedings of IEEE INFOCOM 2004*, Hong Kong, China, March 2004.

[EO82] H. Edelsbrunner and M. H. Overmars. On the equivalence of some rectangle problems. *Information Processing Letters*, 14(3):124–127, 1982.

[Fal86] C. Faloutsos. Multiattribute hashing using gray codes. In *Proceedings of ACM SIGMOD International Conference on the Management of Data*, pages 227–238, 1986.

[Fal88] C. Faloutsos. Gray codes for partial match and range queries. *IEEE Trans. Software Eng.*, 14(10):1381–1393, 1988.

[GG98] V. Gaede and O. Günther. Multidimensional access methods. *ACM Computing Surveys*, 30:170–231, Jun 1998.

[Hil91] D. Hilbert. Uber die stegie abbildung einer linie auf flachenstuck. *Mathematische Annalen*, 38:459–460, 1891.

[LAB+02] V. J. Leung, E. M. Arkin, M. A. Bender, D. P. Bunde, J. Johnston, A. Lal, J. S. B. Mitchell, C. A. Phillips, and S. S. Seiden. Processor allocation on cplant: Achieving general processor locality using one-dimensional allocation strategies. In *Proc. IEEE International Conference on Cluster Computing (CLUSTER)*, pages 296–304, 2002.

[LHJ05] G. Li, S. Hou, and H. Jacobsen. A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams. In *International Conference on Distributed Computing Systems(ICDCS'05)*, 2005.

[MFB02] G. Mühl, L. Fiege, and A. P. Buchmann. Filter similarities in content-based publish/subscribe systems. In *International Conference on Architecture of Computing Systems (ARCS)*, volume 2299 of *Lecture Notes in Computer Science*, pages 224–238, 2002.

[MJFS01] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1), 2001.

[Mor66] G.M. Morton. A computer oriented geodetic data base and a new technique in file sequencing, 1966. IBM,Ottawa,Canada.

[NKV99] A. Nakano, R. K. Kalia, and P. Vashishta. Scalable molecular dynamics visualization, and data management algorithms for materials simulations. *IEEE Computing in Science and Engineering*, 1(5):39–47, 1999.

[OJPA06] Aris Ouksel, Oana Jurca, Ivana Podnar, and Karl Aberer. Efficient probabilistic subsumption checking for content-based publish/subscribe systems. In *Proceedings of ACM/IFIP/USENIX 7th International Middleware Conference, Melbourne, Australia*, December 2006.

[Ora01] Oracle spatial quadtree indexing oracle faq: Oracle spatial data option. `http://www.orafaq.com/faqsdo.htm`, 2001.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.

[SAT05] Z. Shen, S. Aluru, and S. Tirthapura. Indexing for subscription covering in publish-subscribe systems. In *Proceedings of the 18th International Conference on Parallel and Distributed Computing Systems (PDCS)*, September 2005.

[TE04] P. Triantafillou and A. Economides. Subscription summarization: A new paradigm for efficient publish/subscribe systems. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 562–571, 2004.

[TSA06] S. Tirthapura, S. Seal, and S. Aluru. A formal analysis of space filling curves for parallel domain decomposition. In *Proc. IEEE International Conference on Parallel Processing*, 2006. to appear.

[Wil84] D. E. Willard. New trie data structures which support very fast search operations. *J. Comput. Syst. Sci.*, 28(3):379–394, 1984.

[WL85] D. E. Willard and G. S. Lueker. Adding range restriction capability to dynamic data structures. *J. ACM*, 32(3):597–617, 1985.

[WS93] M. Warren and J. Salmon. A parallel hashed-octtree N-body algorithm. In *Proc. Supercomputing*, pages 12–21, 1993.

[ZSB04] Y. Zhao, D. Sturman, and S. Bhola. Subscription propagation in highly-available publish/subscribe middleware. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware (Middleware'04)*, pages 274–293, 2004.

APPENDIX

A Pseudo Code of Section 5 (Algorithm for Approximate Point Dominance)

Due to the typesetting problem, we split the algorithm into three pieces. Algorithm 1 is the main routine, which calls a recursive function *EnumRectangles*. The pseudo code for this routine is listed in Algorithm 3. Inside *EnumRectangles*, we call another recursive function *CompKeys*, whose pseudo code is presented in Algorithm 2.

Algorithm 1 Compute the keys of standard cubes in D_i

```
1: create an empty vector P of size d
2: for j = 1 to d do
3:    if ℓ_{j,i} == 1 then
4:       EnumRectangles(P,j,1)
5:    end if
6: end for
```

Algorithm 2 CompKeys(vector P, vector Q, int t)

Synopsis: compute the keys of standard cubes contained inside the rectangle denoted by P

```
1: use Equation 1 to compute Q_t based on P_t and ℓ_t.
2: for every possible instance of Q_t do
3:    if t == d then
4:       generate the key of a standard cube by interleaving the bits in Q
5:    else
6:       CompKeys(P,Q,t + 1)
7:    end if
8: end for
```

Algorithm 3 EnumRectangles(vector P, int s, int t)

Synopsis: enumerate the rectangles within the space occupied by D_i, whose side length along dimension s is 2^i

```
1: if t > s then
2:    for j = b(ℓ_t) − 1 downto i do
3:       if ℓ_{t,j} == 1 then
4:          P_t = j
5:          if t == d then
6:             create an empty vector Q of size d
7:             CompKeys(P,Q,1)
8:          else
9:             EnumRectangles(P,s,t + 1)
10:         end if
11:      end if
12:   end for
13: else if t < s then
14:   for j = b(ℓ_t) − 1 downto i + 1 do
15:      if ℓ_{t,j} == 1 then
16:         P_t = j
17:         if t == d then
18:            create an empty vector Q of size d
19:            CompKeys(P,Q,1)
20:         else
21:            EnumRectangles(P,s,t + 1)
22:         end if
23:      end if
24:   end for
25: else
26:   P_t = i
27:   if t == d then
28:      create an empty vector Q of size d
29:      CompKeys(P,Q,1)
30:   else
31:      EnumRectangles(P,s,t + 1)
32:   end if
33: end if
```