J. Parallel Distrib. Comput. 72 (2012) 1591–1602

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

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

Zhenhui Shen [a,1], Srikanta Tirthapura [b,*]

[a] Akamai Technologies, 8 Cambridge Center, Cambridge, MA, 02142, USA

[b] Department of Electrical and Computer Engineering, Iowa State University, 2215 Coover Hall, Ames, IA, 50010, USA

ARTICLE INFO

Article history:
Received 22 September 2011
Received in revised form
24 July 2012
Accepted 5 September 2012
Available online 17 September 2012

Keywords:
Publish-subscribe
Subscription covering
Space filling curve

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.

1. Introduction

Content-based publish–subscribe systems offer an expressive middleware to distributed applications. Such systems route messages from senders (publishers) to receivers (subscribers) based on the message content, rather than on a destination name or address. In order to express their personal interest, a subscriber provides appropriate *subscriptions* to the system, where each subscription is a predicate on the message content. A subscriber is delivered only those messages which match at least one of its subscriptions. For example, the event [location = Ames, temp = 12, light intensity = 50] must be delivered to a subscriber who has issued the subscription [location = Ames, temp < 30, light intensity < 75]. Content-based routing is an essential middleware for many modern distributed applications. Since a centralized implementation of publish–subscribe may not scale to large networks, there have been many efforts on building content-based publish–subscribe on a distributed network of routers (or "brokers"), e.g. Gryphon [28], Siena [1], JEDI [5], REBECA [16], and XNET [2].

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 forward each subscription to every router in the network. However, such an approach can lead to increased network traffic as well as a larger size of routing tables, making event forwarding less efficient. A better approach, that has been advocated and used by multiple groups [13,1,24,21], 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 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 many earlier systems, including [1,5,16,2].

However, identifying covering relationships among subscriptions is itself a hard combinatorial problem, for which no solutions are known that are time-efficient in the worst case. 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

☆ A preliminary version of this work appeared in the Proceedings of the 27th IEEE International Conference on Distributed Computing Systems (ICDCS 2007).

* Corresponding author.
E-mail addresses: zshen@akamai.com (Z. Shen), snt@iastate.edu (S. Tirthapura).

[1] Part of the work done while a student in the Department of Electrical and Computer Engineering, Iowa State University.

0743-7315/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2012.09.002

time and space complexity [3,4], showing that there does not exist a worst-case time efficient solution. This is an instance of 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.

We observe that in the 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. In this paper 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 that is searched for covering subscriptions. 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 an index for approximate covering detection 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 [8], data partitioning in parallel computers [12], and in scientific computing [25,17]. However, as explained above, SFCs (or any other spatial data structure, for that matter) do not provide worst-case time efficient solutions to exact point dominance in high dimension, 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 [6] 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 [6] 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 along 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 $\alpha = b_{\max} - b_{\min}$.[2] 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 [18,9]. Other popularly used SFCs are the Hilbert curve [10] and the Gray code curve [7]. It has been observed [14] 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 $O[\log \frac{d}{\epsilon} \cdot (2^{\alpha+1}\frac{d}{\epsilon})^{d-1}]$.
2. In contrast, the worst case time complexity of an exhaustive point dominance query using the Z SFC is $\Omega[(2^{\alpha-1}\ell)^{d-1}]$ 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.

[2] 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 the aspect ratio is approximately the logarithm (to base 2) of the traditional definition. This definition leads to a convenient statement of our results.

4. We present a simulation study showing the practical performance of an approximate search for different input dimensions, aspect ratios, and space coverage ratios.

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 largest side length of the query region. This implies that for a constant ϵ, an ϵ-approximate query is much cheaper than an exhaustive query, if α is small. Further, we can expect that the benefits of approximate covering over exhaustive covering will be more pronounced as the query region gets larger.

Effect of the number of dimensions. If the number of dimensions is increased, but the aspect ratio is the same, the worst case complexity increases, since the theoretical result on the cost of approximate searching has the term d^{d-1}. Note that the cost of exact searching also increases as ℓ^{d-1} where ℓ is the size of the smallest side of the query region, but overall, the speedup due to approximate searching decreases as the number of dimensions increases. Hence, the speedup due to approximate covering is smaller for larger dimensions, as was also observed by our experimental study.

Effect of the aspect ratio. 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. If however, the aspect ratio of the query rectangle is 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. As the value of ℓ, the shortest side length decreases, the aspect ratio will increase, and the advantage of approximate covering becomes smaller.

Simulation study. We present a simulation study that compares the cost of an exact search algorithm with the cost of an approximate search, for different input dimensions, aspect ratios, and space coverage ratios. Overall, while our theoretical results are all asymptotic in nature, and hence apply for the case when the size of the query regions and the size of the universe are very large, our simulations show that there is a real advantage to be gained in practical scenarios through the use of approximate search. The key observation is that *if it is required to search only an δ fraction of the search space, it is typically possible to attain a speedup of much more than $1/\delta$*.

1.3. Related work

Tran and Nguyen [23] propose an alternate approach to approximate subscription covering. In their approach, the main optimization is that given a new subscription s_1 and an existing subscription s_2, an approximate check of whether or not s_2 covers s_1 is done in time sublinear in d, the number of dimensions in the event space. But the time taken for searching the set of all current subscriptions is still linear in n, the number of subscriptions, in the worst case. This approach is suitable for subscriptions that have a large number of attributes. In contrast, our method searches a large fraction (but not all) of the space of current subscriptions, by using appropriate indexes for which approximate search is fast, so our method uses time sublinear in the number of current subscriptions.

Ouksel et al. [19] consider a more general problem of checking whether a given new subscription is covered by the *union* of all existing subscriptions. They propose a probabilistic solution to the above problem, with complexity $O(nd)$, where n is the number of subscriptions and d is the dimension of the event space. In contrast, we consider the problem of checking whether there is a single existing subscription that completely covers the arriving subscription. The complexity of our solution is sublinear in n.

Jafarpour et al. [11] present an approach to organizing subscriptions and data using space filling curves. The main difference between their work and ours is that we have addressed in detail the cost of an approximate search for covering subscriptions, and treat this analytically in a detailed manner. In contrast, their work does not address the issue of approximate covering in detail.

From a worst-case time complexity perspective, the current best solution to point-dominance problem (see [20, Chapter 8], [26,27]) 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.

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. [14] 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. [22] present a formal analysis of the performance 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 describe our algorithm for approximate point dominance in Section 5, present our simulation results in Section 6, and conclusions in Section 7.

2. Space filling curves

We consider a d-dimensional universe $\mathcal{U}$ of size $2^k \times 2^k \cdots \times 2^k$. Note that the number of dimensions d is twice the number of attributes in a subscription. If different attributes are chosen from domains of different sizes, we will need to normalize them to be of the same size. Each element of $\mathcal{U}$ is a point $p = (x_1, x_2, \ldots, x_d)$, where for $i \in [1, d]$, $x_i \in [0, 2^k - 1]$. A space filling curve is a linear order on all 2^{kd} points in $\mathcal{U}$. In the rest of this paper, we use the terms "point" and "cell" interchangeably. Henceforth we use the term "cube" to refer to a cube in d dimensions, whose side length is the same along every dimension, and "rectangle" to refer to a rectangle in d dimensions.

Most popular SFCs, including the Z curve [15] and the Hilbert curve [10] use a recursive partitioning of the universe. The universe $\mathcal{U}$ is first divided into 2^d cubes, each of side length 2^{k-1}, by bisecting $\mathcal{U}$ along each dimension. Each resulting cube is recursively divided into 2^d cubes in the same manner, and the recursion continues until a depth of k, at which point we are left with cubes with a side length of 1. This recursive subdivision of the universe can be viewed as a rooted tree where each node in the tree is a cube in d-dimensions, the root is the entire universe $\mathcal{U}$, and the children of a node v are all the 2^d cubes derived from the cube v by bisecting v along every dimension.

We use the term *standard cube* to refer to each cube that is a node in the above tree. Note that not every cube in $\mathcal{U}$ is a standard

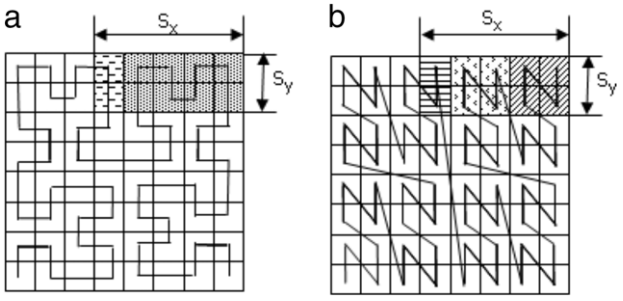


Fig. 1. For the same $S_x \times S_y$ rectangle, there are (a) Two runs for the Hilbert SFC and (b) Three runs for the Z SFC.

cube. When $\mathcal{U}$ 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. A standard cubes at level k is an individual cell.

Lemma 2.1. *Let C_1 and C_2 be two standard cubes which are not equal to each other. Then either C_1 contains C_2, or C_2 contains C_1, or C_1 and C_2 are disjoint from each other.*

Proof. Consider the tree corresponding to the recursive partitioning of $\mathcal{U}$. Both C_1 and C_2 are nodes in this tree. Since $C_1 \neq C_2$, the following are the only possibilities (1) C_1 is an ancestor of C_2, or (2) C_2 is an ancestor of C_1, or (3) there is no overlap between the subtrees rooted at C_1 and C_2. In the first case, C_1 contains C_2, in the second case C_2 contains C_1 and in the third case C_1 and C_2 are disjoint from each other. □

Each edge in the above tree is given an integer d-bit label such that the labels of edges from a parent to its children are unique (each internal node has 2^d children). Different SFCs, such as the Z curve and the Hilbert curve differ in the way these labels are assigned to the different children of a node. Each standard cube at level $\ell \leq k$ is assigned a unique $d \cdot \ell$ bit number called its *key*, which is formed by concatenating the labels of all the on the (unique) length ℓ path from the root to the cube. Thus every cell also has a key, which is a bit string of length kd.

The SFC is used to organize the points into an one-dimensional data structure, that we call the *SFC array*, as follows. The input points are sorted according to the keys of the cells containing them, and stored in an array. Note that the SFC array could be implemented using an array. If it is required to support the dynamic addition and deletion of points, a dynamic data structure such as a binary tree or a skip list can be used. The SFC array is the only data structure that we need for further use.

A *run* is defined as a set of cells that are consecutively ordered by the SFC. For example, for the rectangular region shown in Fig. 1, the Z SFC will lead to three runs, and the Hilbert SFC will lead to two runs. All cells belonging to a run appear as a contiguous segment in the SFC array.

Accessing a run in the SFC array requires two binary searches on the keys corresponding to the first and last cells in the run. Hence, certain operations on a run, such as examining if a run is empty or not, or determining the size of the run, are very efficient, no matter whether the run is small or large. In our case, the only query we make on the SFC is whether or not a run is empty, and hence, the performance of the SFC for the 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, such as the Hilbert curve. 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 cube is a single run.*

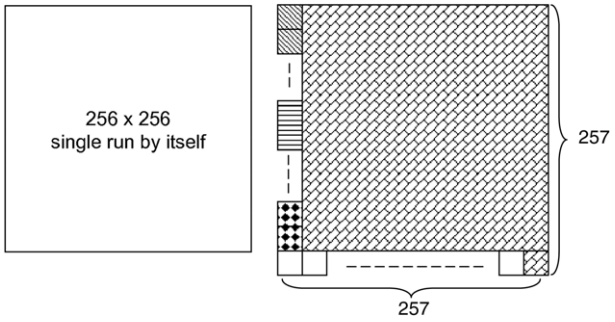


Fig. 2. Two example point dominance queries for the Z curve. Standard cubes belonging to the same run are drawn using identical patterns.

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. Handling a single run, no matter whether large or small, is inexpensive, since the typical operation related to a run is to see if a run is empty or not, which can be done in time logarithmic in the number of subscriptions if all points are sorted according to their position in the SFC. Accessing 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. Fig. 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. However, for the second query region, one of the runs covers more than 99% of the query region, while all the other runs together cover only 1% 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 search, in this case.

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 $p = (x_1, x_2, \ldots, x_d)$, exhaustive point dominance searches for a point in the 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, if $\ell = (\ell_1, \ell_2, \ldots, \ell_d)$, then $t(\ell, m) = (t(\ell_1, m), t(\ell_2, m), \ldots, t(\ell_d, m))$.

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, let cubes(T) denote the minimum number of standard cubes into which T can be partitioned, and runs(T) denote the minimum number of runs of the Z SFC, whose union is equal to all cells of T in the SFC array. Let vol(T) denote the number of cells in T.

The following observation follows since each standard cube is a single run.

Observation 3.1. *For any rectangle T,* runs$(T) \leq$ cubes(T).

Note that it is possible that runs$(T) <$ cubes(T). The worst-case cost of approximate point dominance is equal to runs$(R^m(\ell))$. Using Observation 3.1, this is bounded by cubes$(R^m(\ell))$. In the following, we will prove an upper bound for cubes$(R^m(\ell))$, using the following strategy. Lemma 3.2 shows that a greedy algorithm can partition an arbitrary region into a minimum number of standard cubes. Lemma 3.3 classifies the obtained standard cubes according to their sizes; this lets us compute the total number of cubes of each size. Finally, these are combined in Lemma 3.6 to yield an upper bound on the total number of runs that make up any query region. Note that our proofs for the upper bound depend solely on bounding cubes(T), so these apply to any other space filling curve that is based on a similar recursive decomposition of the universe, such as the Hilbert curve.

As the first step, we decide what value of m such that $R^m(\ell)$ has 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 3.1. *Let $0 < \epsilon < 1$. If $m \geq \log_2 \frac{2d}{\epsilon}$, then $\frac{\text{vol}(R^m(\ell))}{\text{vol}(R(\ell))} \geq 1 - \epsilon$.*

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

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

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

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 subset of $\mathcal{U}$ into standard cubes. Note that it is trivial to partition any region into a set of standard cubes, since the individual cells themselves are standard cubes. We now describe a greedy algorithm that outputs the partition of any region R (which is not necessarily a rectangle) into a *minimum* number of standard cubes. The greedy algorithm works as follows. *If the region 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 3.2. *The greedy algorithm, when applied to region R, produces an optimal partition of R into a minimum number of standard cubes.*

Proof. We prove the result 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 E 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 a 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 $D(\ell)$ represent the set of standard cubes resulting from a greedy decomposition of $R(\ell)$. Let $D_i(\ell)$ represent a subset of $D(\ell)$ consisting of standard cubes of side length 2^i. For integer x, let x_j denote the jth bit in x, where x_0 is the least significant bit of x and $x_{b(x)-1}$ is the most significant bit of x.

Let $S_i(x) = t(x, b(x) - i)$ denote the result of retaining only the most significant bits in x starting from x_i onwards, and zeroing out the rest. When the input is a vector, the operator $S_i()$ will be applied to each element in the vector. Thus $S_i(\ell) = (S_i(\ell_1), S_i(\ell_2), \ldots, S_i(\ell_d))$.

For $i = 1 \cdots d$ and $j = 0 \cdots b(\ell_i) - 1$, let $\ell_{i,j}$ denote $(\ell_i)_j$. For $j \in [0, k]$, indicator variable $O_j(\ell)$ is defined as follows: $O_j(\ell) = 0$, if $\ell_{i,j} = 0$ for all $i \in [1, d]$; $O_j(\ell) = 1$, if $\ell_{i,j} = 1$ for some $i \in [1, d]$. Assume w.l.o.g. $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_d$. The following lemma characterizes the type and location of the standard cubes resulting from an optimal partition of $R(\ell)$.

Lemma 3.3. *For $b(\ell_1) \leq i \leq b(\ell_d) - 1$, $D_i(\ell)$ is empty. For $0 \leq i \leq b(\ell_1) - 1$*

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

Proof. First we prove by contradiction that $D_i(\ell)$ is empty for $b(\ell_1) \le i \le b(\ell_d) - 1$. Suppose $D_j(\ell)$ is non-empty for some j, $b(\ell_1) \le j \le b(\ell_d) - 1$. Let $C_j \in D_j(\ell)$ 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 $2^j \ge 2^{b(\ell_1)}$. But, the projection of the cube along the first dimension can be no more than ℓ_1, which is at most $2^{b(\ell_1)} - 1$. This is a contradiction. Thus $D_i(\ell)$ must be empty for $b(\ell_1) \le i \le b(\ell_d) - 1$.

For i such that $0 \le i \le b(\ell_1) - 1$, 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 $O_{b(\ell_1)-1}(\ell) = 1$, 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 $1 \le j \le d$, 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 $1 \le j \le d$. 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 $D_i(\ell)$ is empty for $b(\ell_d) - 1 \ge i \ge b(\ell_1)$. 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 $D_{b(\ell_1)-1}(\ell)$ 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 $C \in D_{b(\ell_1)-1}(\ell)$. 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 $R(\ell) - R(S_{b(\ell_1)-1}(\ell))$ cannot contain C. As a consequence, the region occupied by $D_{b(\ell_1)-1}(\ell)$ 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, $b(\ell_1) - 1 \ge i \ge \kappa + 1$. We now consider the case $i = \kappa$. We study the following two cases.

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

Case II: $O_\kappa(\ell) = 1$. Consider the extremal rectangles $R(S_\kappa(\ell))$ and $R(S_{\kappa+1}(\ell))$. Since $S_{\kappa+1}(\ell_i) \le S_\kappa(\ell_i) \le \ell_i$, 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 $1 \le i \le d$, 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 $O_\kappa(\ell) = 1$, 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 $D_\kappa(\ell)$ is non-empty and the union of standard cubes of side length 2^κ or higher in $D(\ell)$ form the extremal rectangle $R(S_\kappa(\ell))$. □

Lemma 3.3 can be used to totalize the minimum number of standard cubes in $R^m(\ell)$ as stated in Lemma 3.6.

We will need the following additional definition and lemma for the proof of Lemma 3.5. For every $i = 0, \ldots, b(\ell_1) - 1$, let $N_i = |D_i(\ell)|$.

Lemma 3.4. *For every $i = 0, \ldots, b(\ell_1) - 1$, we have the following. If $O_i(\ell) = 0$, then $N_i = 0$. If $O_i(\ell) = 1$, 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 3.3, we know that if $O_i(\ell) = 0$, then $N_i = 0$. If $O_i(\ell) = 1$, all standard cubes in $D_i(\ell)$ 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 (which in turn can be obtained through Lemma 3.3) by the volume of a standard cube of side length 2^i. □

Lemma 3.5. $\text{cubes}(R^m(\ell))$ *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 3.3, we have $\text{cubes}(R^k(\ell)) = \sum_{i=b(\ell_1)-k}^{i=b(\ell_1)-1} N_i$. Using Lemma 3.4,

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

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. □

Lemma 3.6. $\text{cubes}(R^m(\ell)) < m \cdot [2^\alpha(2^m - 1)]^{d-1}$.

Proof. We consider two cases: (1) $k < \alpha$ and (2) $k \ge \alpha$. To find an upper bound on $\text{cubes}(R^k(\ell))$, we assume vector ℓ satisfies the conditions specified by Lemma 3.5.

Case 1: $k < \alpha$.

According to Lemma 3.4, 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 3.5, we have $\ell_{j,x} = 1$ for $x \in [b(\ell_d) - k, b(\ell_d) - 1]$ and $1 < j \le d$.

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 $1 < j \le d$.

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

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

Case 2: $k \ge \alpha$.

According to Lemma 3.4, 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 3.5, we have $\ell_{j,x} = 1$ for $x \in [b(\ell_d) - k, b(\ell_d) - 1]$ and $1 < j \le d$.

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

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

Note that

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

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 $X \approx 2^\alpha \cdot Y$.

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

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

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

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.

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 $O(\log \frac{d}{\epsilon} \cdot (2^{\alpha+1} \frac{d}{\epsilon})^{d-1})$.*

Proof. Using Lemma 3.6 and since $\text{runs}(R^m(\ell)) \le \text{cubes}(R^m(\ell))$ (Observation 3.1), we have $\text{runs}(R^m(\ell)) < m \cdot [2^\alpha (2^m - 1)]^{d-1}$. From Lemma 3.1, 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. $\square$

4. Lower bound for exhaustive point dominance

In this section, we prove a lower bound on the worst-case cost of exhaustive point dominance using the Z-curve (Theorem 4.1). The worst-case cost is equal to $\text{runs}(R(\ell))$. However, estimating the size of $\text{runs}(R(\ell))$ is much harder than estimating the size of $\text{cubes}(R(\ell))$, because there is no simple characterization for $\text{runs}(R(\ell))$, similar to the "greedy" characterization for $\text{cubes}(R(\ell))$.

Our strategy for proving a lower bound is as follows. We construct an extremal rectangle $R(\ell)$ such that for a large subset of the standard cubes resulting from a greedy (optimal) partition of $R(\ell)$, no two cubes in the subset can belong to the same run in the Z curve. Thus $\text{runs}(R(\ell))$ is no less than the size of this subset of standard cubes.

Let γ be an integer in $(0, k - \alpha]$. Given a value of the aspect ratio α, we consider the following extremal rectangle $R(\ell)$: (1) $\ell_d = 2^\gamma - 1$ and (2) $b(\ell_i) = \gamma + \alpha$ for $i \in [1, d)$. We examine a smaller rectangle R_0 contained inside $R(\ell)$. R_0 is constructed by selecting the least significant bit from ℓ_d and the most significant bit from ℓ_i for $i \in [1, d)$. Thus, the side length of R_0 along dimension d is 1, and the side length of R_0 along all other dimensions is $2^{b(\ell_1)-1}$. When the greedy partition is applied, R_0 is filled with standard cubes with a side length of 1, since its shortest side has length 1.

Note that every standard cube in rectangle R_0 must be a cell, since one side of R_0 has unit length. The coordinates of each cell in R_0 exhibits the pattern shown in Fig. 3, and the key of the cell is formed by interleaving the bits representing its coordinates, as shown Fig. 4. Lemma 4.1 uses this pattern to prove that no two cells in R_0 belong to the same run on the Z-curve.

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

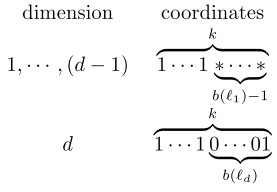


Fig. 3. Coordinates of a cell in R_0. Symbol $*$ can be either 1 or 0. The size of the universe along each dimension is 2^k.

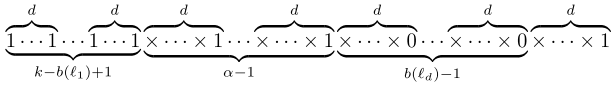


Fig. 4. Key of a cell in R_0.

Proof. 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 Fig. 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 Fig. 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)

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

5. Algorithm for approximate point dominance

In this section, we present an algorithm for approximate point dominance based on the Z SFC. The input points are organized into an SFC array (described in Section 2), in which they are sorted according to their positions on the Z curve. It is 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. It then searches these cubes in the SFC array for covering subscriptions, in the descending order of their volumes. Meanwhile it keeps track of the ratio of the volume searched to the volume of the query region. The search terminates when either a covering subscription is found or this ratio exceeds $1 - \epsilon$.

Let $R(\ell)$ denote the extremal rectangle to be searched for a point dominance query, where $\ell = (\ell_1, \ell_2, \ldots, \ell_d)$. Recall that $D_i(\ell)$ (defined in Section 3) represents the set of standard cubes resulting from the greedy decomposition of $R(\ell)$, whose side length is 2^i. Our algorithm will search the standard cubes in $R(\ell)$ in the descending order of i. In what follows, we show how to enumerate the standard cubes within $D_i(\ell)$ when it is not empty.

The enumeration is a 2-step recursive process. Let $A(D_i(\ell))$ represent the area occupied by $D_i(\ell)$. We first divide $A(D_i(\ell))$ into a set of disjoint rectangles, each of which is a union of standard cubes within $D_i(\ell)$. The rectangles are classified into categories according to their side lengths, as follows. For $1 \leq m \leq d$, let $R_{i,m}(\ell)$ denote the set of rectangles that satisfy the following two properties: (1) The side length of the rectangle along m dimensions is exactly 2^i (2) The side length of the rectangle along the remaining $(d - m)$ dimensions is a multiple of 2^i. The following equation follows from the definition of $R_{i,m}$.

$$A(D_i(\ell)) = \bigcup_{m=1}^{m=d} R_{i,m}(\ell).$$

We first enumerate all rectangles within $E_{i,m}(\ell)$ and then the standard cubes contained within each resulting rectangle. First, we demonstrate how to enumerate the rectangles within $R_{i,m}(\ell)$, if it is not empty. Let vector $r(\ell, i, m)$ express a rectangle within $R_{i,m}(\ell)$. The size of $r(\ell, i, m)$ is d. Element $r[j]$ records the position of a non-zero bit in ℓ_j. Furthermore $2^{r[j]}$ measures the side length of the rectangle along dimension j. According to the definition of $R_{i,m}(\ell)$, there are exactly m elements in $r(\ell, i, m)$ satisfying $r[j] = i$. For the remaining $(d - m)$ elements, we have $r[j] \in [i + 1, b(\ell_j) - 1]$. Let $b(x)$ represent the number of bits in the binary representation of integer x.

Let NonZeroDim(ℓ, i) denote all the dimensions j that satisfy $\ell_{j,i} = 1$. Let $\ell_{j,i}$ denotes the ith bit of ℓ_j. Let FixedDim(ℓ, i, m) denote all unique combinations of m elements in NonZeroDim(ℓ, i). If $R_{i,m}(\ell)$ is not empty, the size of NonZeroDim(ℓ, i) is no less than m. So FixedDim(ℓ, i, m) is also non-empty.

Let $fd(\ell, i, m)$ be an element in FixedDim(ℓ, i, m). We can use it to settle the value of m elements in $r(\ell, i, m)$. In particular, for the m dimensions characterized by $fd(\ell, i, m)$, the corresponding $r[j]$ is equal to i. For the remaining $(d - m)$ dimensions, the choices for the corresponding $r[j]$ = are the positions of non-zero bits in ℓ_j between $i+1$ and $b(\ell_j)-1$. So it may not be limited to a single value. Every unique combination of these $(d - m)$ elements refers to a unique rectangle in $R_{i,m}(\ell)$, subject to the restriction of $fd(\ell, i, m)$. Algorithm 1 describes a procedure to return a different instance of $r(\ell, i, m)$ upon each invocation, subject to the restriction of $fd(\ell, i, m)$.

We use an example to explain Algorithm 1. Suppose we are given an extremal rectangle 7×5. The picture shows the space occupied by $D_0(\ell)$, i.e. the union of standard cubes with side length equal to 1. We notice $A(D_0(\ell))$ can be divided into 4 disjoint rectangles (see Fig. 5). Let $r = (r_1, r_2)$ be the expression of a such rectangle. Fig. 6 reveals the connection between $\ell = (\ell_1, \ell_2)$ and $r = (r_1, r_2)$. If we fix dimension 1, we have $r = (0, 2)$. This refers to rectangle 1×4. If we fix dimension 2, we have $r = (1, 0)$ or $r = (2, 0)$. That gives us two rectangles 2×1 and 4×1. If we fix both dimensions, we have $r = (0, 0)$. That returns us the last rectangle 1×1.

As the second step, we demonstrate how to enumerate the standard cubes contained within a rectangle represented by vector

Algorithm 1: EnumerateRectangle

```
/* This procedure returns a different
   rectangle within R_{i,m}(ℓ) upon each
   invocation, subject to the constraint of
   fd(ℓ, i, m). The notations used in the
   algorithm are defined in Section 5.     */
```

Input:

$\ell = (\ell_1, \ell_2, \ldots, \ell_d)$: the side lengths of an extremal rectangle;
i: The returned rectangle is a union of standard cubes with side length equal to 2^i;
m : The total number of dimensions, along which the side length of the returned rectangle is 2^i;
$fd(\ell, i, m)$: an element in FixedDim(ℓ, i, m);
initFlag: Initialization is required, if this flag is set to true;

Output:

$r(\ell, i, m)$: a vector of size d describing a rectangle within $R_{i,m}(\ell)$. Element $r[j]$ records the position of a non-zero bit in ℓ_j. Furthermore $2^{r[j]}$ measures the side length of the returned rectangle along dimension j. If the returned value is $(-1, -1, \ldots, -1)$, it indicates the end of enumeration;

```
begin
    Create two vectors of size d : NextRectangle and
    NoMoreRectangle. They are declared as static variables to
    the procedure;
    if initFlag is true then
        NoMoreRectangle ⇐ (−1, −1, ..., −1);
        for z from 1 to d do
            if z ∈ fd(ℓ, i, m) then
                NextRectangle[z] ⇐ i;
            else
                NextRectangle[z] ⇐ b(ℓ_z) − 1;
        return NextRectangle;
    for z from 1 to d do
        if z ∉ fd(ℓ, i, m) then
            NextRectangle[z] ⇐ the position of the next
            non-zero bit in ℓ_z between NextRectangle[z] − 1
            and i + 1;
            if NextRectangle[z] is undefined then
                NextRectangle[z] ⇐ b(ℓ_z) − 1;
            else
                return NextRectangle;
    return NoMoreRectangle;
    /* If we finish the entire for loop,
       NextRectangle rotates back to the
       vector when it first gets initialized.
       This indicates the termination of the
       enumeration.                          */
```

$r(\ell, i, m)$. We use vector $c(\ell, i, m)$ to represent a such cube. The vector has a size of d. These standard cubes have a side length of 2^i. They are obtained after $(k - i)$ rounds of recursive partitioning of the universe as described in Section 2. When the recursion stops at the depth of $(k - i)$, each dimension of the universe is split into $2^{(k-i)}$ number of equal line segments. Furthermore a cube's projection on any dimension is one of the resulting line segments. We sequentially assign each line segment a unique index number (starting at 0 from the low end). The index number is an integer of $(k-i)$ bits. Element $c[j]$ in $c(\ell, i, m)$ records the index number of a

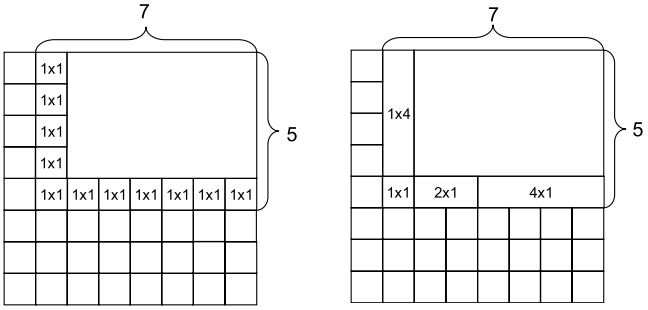


Fig. 5. Enumeration of rectangles that are the union of standard cubes with side length equal to 2^0.

cube's projection on dimension j. Thus vector $c(\ell, i, m)$ defines the cube's position in the universe.

Notation $r[j]$ represents an element in $r(\ell, i, m)$, and $c[j]$ represents an element in $c(\ell, i, m)$. Let $\ell_{j,x}$ denote the xth bit of ℓ_j, and $c[j]_x$ denote the xth bit of $c[j]$. Section 2 defines the size of the d-dimensional universe $\mathcal{U}$ to be $2^k \times 2^k \cdots \times 2^k$. The value of $c[j]$ can be derived from ℓ_j and $r[j]$ as follows.

$$\begin{aligned} c[j]_{x-i} &= \neg \ell_{j,x}, && \text{when } x \in [r[j] + 1, k - 1] \\ c[j]_{x-i} &= 1, && \text{when } x = r[j] \\ c[j]_{x-i} &= 0 \quad \text{or } 1, && \text{when } x \in [i, r[j] - 1]. \end{aligned} \tag{1}$$

Since the lower bits of $c[j]$ (between the position 0 and $r[j]-i-1$) are not determined, $c[j]$ may not be limited to a single value. It is not difficult to see that every unique combination of the possible values of $c[1]$ through $c[d]$ defines a unique instance of $c(\ell, i, m)$. Algorithm 2 describes a procedure to enumerate all the instances of $c(\ell, i, m)$.

We continue using the previous example to explain Algorithm 2. Rectangle 2×1 in $A(D_0(\ell))$ contains two standard cubes. The corresponding vector $r = (1, 0)$. We use $c = (c_1, c_2)$ to express a standard cube contained within r. Based on Eq. (1), the possible values of c_1 are $(010)_2$ and $(011)_2$, c_2 has only one value $(011)_2$. The combination of $((010)_2, (011)_2)$ describes the left standard cube, and the combination of $((011)_2, (011)_2)$ gives the right standard cube.

Once we have the expression of $c(\ell, i, m)$, it is easy to search the corresponding cube in the SFC array. We need to compute the keys of the first and last cells in the cube. Each key is an integer of $d \times k$ bits. The $d \times (k - i)$ bits in the highest order of the key can be obtained by interleaving the bits of $c[1]$ through $c[d]$ in the order from the highest to the lowest. The remaining $d \times i$ bits in the lowest order of the key are all 0 for the first cell, and all 1 for the last cell.

6. Simulation study

In this section, we report on the results from a simulation study of the algorithm for approximate covering detection. We evaluate the cost of an approximate search, as well as the advantage in using approximate search versus exhaustive search, using the Z space filling curve, and the algorithm as described in Section 5. This evaluation is done for different number of dimensions, and for different aspect ratios. We considered the most expensive cases of searching of "sparse" presence of covering subscriptions, where the entire search region has to be searched to find a covering subscription, and the search cannot stop early. This models the case when either there is no covering subscription, or there are very few covering subscriptions, so that most of the cells in the feasible region have to be examined to find a covering subscription. Other cases of "dense" presence of covering subscriptions are less interesting in our context, since in such cases, even an exhaustive

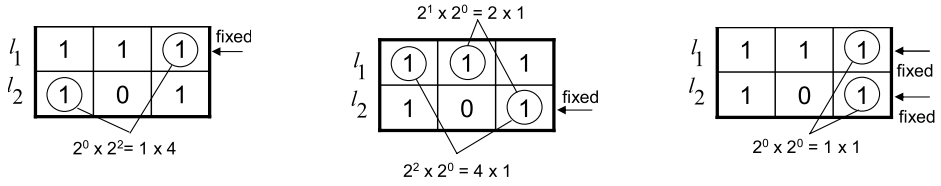


Fig. 6. Enumeration of rectangles that are the union of standard cubes with side length equal to 2^0.

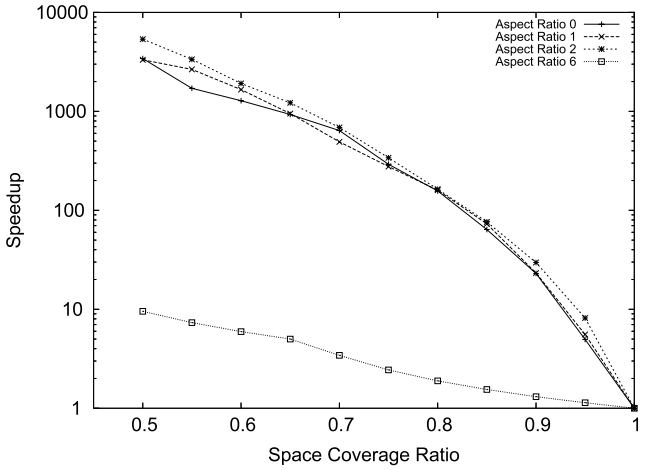


Fig. 7. Speedup through approximate search, dimension = 4.

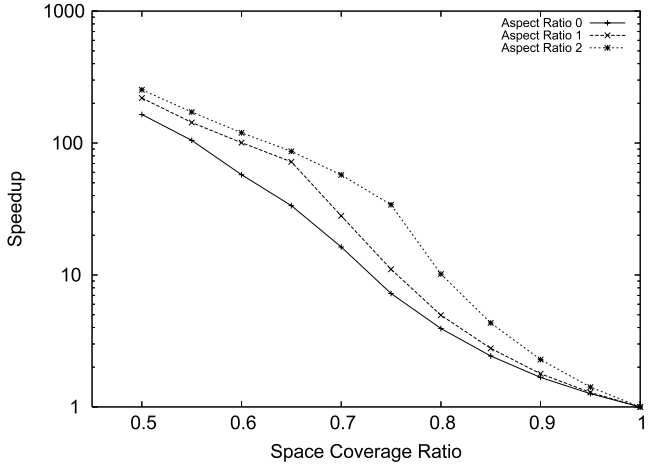


Fig. 8. Speedup through approximate search, dimension = 6.

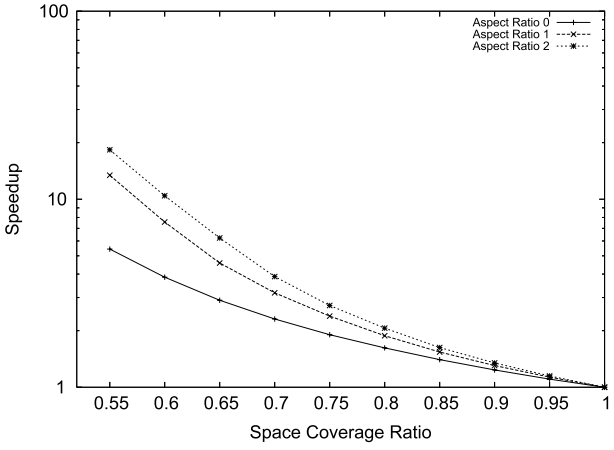


Fig. 9. Speedup through approximate search, dimension = 8.

search can stop early soon after a covering subscription has been found. Further, such cases of dense presence of covering subscriptions do not lead to expensive search times, so that it may not be necessary to optimize any further.

If the rectangle in a d dimensional universe is restricted to have an aspect ratio of α, this poses constraints on the lengths of different sides of the rectangle. We generate a query rectangle by selecting the coordinates of one of the endpoints of the rectangle randomly, subject to the constraint that the shorter side length is still allowed in the universe, then fixing the length of the shorter side, and then choosing the lengths of the remaining sides in accordance with the constraint that no side can be longer than that governed by the aspect ratio. For each point in the plot below, we considered the mean search time for 50 different queries, generated randomly subject to the above constraints.

The following plots show the cost of an exhaustive search for covering subscriptions, by examining the entire search space for a dominating point, in comparison with the cost of an approximate search for covering subscription. The "space coverage" parameter is the ratio of the volume of the space searched by the approximate search to the total volume of the space to be searched for a covering subscription, and is shown on the x-axis. There are two types of plots, one for the query time for 50 approximate search queries (shown on the y-axis), and the other where the speedup of approximate search shown on the y-axis (the speedup is defined as the ratio between the time for exhaustive versus approximate search).

The "dimension" parameter is the number of dimensions of the universe where the search a dominating subscription is conducted. Note that this is twice the number of dimensions in the original data points. For instance, if the event space is two-dimensional (i.e. two attributes), the space for point dominance search is a four dimensional space (see Figs. 7–9).

Discussion. For every case studied, approximate search yielded a speedup which was greater than the inverse of the space coverage ratio. In other words, as expected, the cost of searching say, 65% of the search space (space coverage = 0.65) is much less than 65% of the cost of searching 100% of the search space. Typically, the speedups were much more than the inverse of the space coverage ratio. For instance, with 4 dimensions, and aspect ratio less than 0, 1, or 2, the speedups for 70% coverage was nearly a factor of 500. With 4 dimensions, and an aspect ratio of 0, 1, or 2, the speedup for 90% coverage is nearly a factor of 20. The speedup is much smaller for larger dimensions (dimension = 8) but is still significant, especially for coverage ratios that are 0.75 or smaller. For instance, with 8 dimensions, and a coverage ratio of 0.75, the speedup was nearly threefold, for aspect ratios 0, 1, and 2.

The decrease in speedup with the number of dimensions is consistent with our theoretical observations, as described in Section 1.2. Also to note is that as the extremal rectangle becomes larger (with the aspect ratio the same), the speedup due to approximate searching becomes larger, since large search rectangles are more likely to contain large runs completely within them. In general, the speedup decreases as the aspect ratio increases, sometimes substantially so.

Algorithm 2: EnumStdCube

```
/* This procedure returns a different
   standard cube contained inside the
   rectangle denoted by r(ℓ, i, m) upon each
   invocation. The notations used in the
   algorithm are defined in Section 5.     */
```

Input:

$r(\ell, i, m)$: a vector of size d describing the input rectangle;
k: The size of the d-dimensional universe $\mathcal{U}$ is $2^k \times 2^k \cdots \times 2^k$;
$\ell = (\ell_1, \ell_2, \ldots, \ell_d)$: the side lengths of an extremal rectangle;
i: The side length of the returned standard cube is 2^i;
m : The total number of dimensions, along which the side length of the input rectangle is 2^i;
initFlag: Initialization is required, if this flag is set to true;

Output:

$c(\ell, i, m)$: a vector of size d describing the returned standard cube. Element $c[j]$ records the position of the cube's projection on dimension j. If the returned value is $(-1, -1, \ldots, -1)$, it indicates the end of enumeration;

```
begin
    Create two vectors of size d : NextCube and NoMoreCube.
    They are declared as static variables to the procedure;
    if initFlag is true then
        NoMoreCube ⇐ (−1, −1, ..., −1);
        for z from 1 to d do
            for x from i to k − 1 do
                if x ∈ [i, r[z] − 1] then
                    NextCube[z]_{x−i} ⇐ 0;
                else if x ∈ [r[z] + 1, k − 1] then
                    NextCube[z]_{x−i} ⇐ ¬ℓ_{z,x};
                else
                    NextCube[z]_x ⇐ 1;
        return NextCube;
    for z from 1 to d do
        Create temp as a bit vector of size r[z] − i;
        temp ⇐ the lower bits of NextCube[z] between the
        position 0 and r[z] − i − 1;
        if temp is defined then
            temp ⇐ temp + 1;
            Set the lower bits of NextCube[z] between the
            position 0 and r[z] − i − 1 to temp;
            if temp > 0 then
                return NextCube;
    return NoMoreCube;
    /* If we finish the entire for loop,
       NextCube rotates back to the vector
       when it first gets initialized. This
       indicates the termination of the
       enumeration.                          */
```

Overall, while our theoretical results are asymptotic in nature and hence apply for the case when the size of the query regions and the size of the universe are very large, our simulations show that there is a real advantage to be gained in practical scenarios through the use of approximate search. The key observation is that if it is required to search only an δ fraction of the search space, it is possible to attain a speedup of much more than $1/\delta$.

7. Conclusion

Detection of covering among content-based subscriptions is a hard combinatorial problem. Previous work on this problem has focused on *exact* algorithms for covering detection. In this work we observe that subscription covering is just an optimization, which does not need to be strictly followed all the time. Based on this observation, we introduce the notion of *approximate* covering, which allows us to retain most of the benefits of exact covering at a fraction of its cost. Our approximate solution is based on space filling curves and supports numeric subscriptions. Our analysis shows that when the aspect ratio of the query region is small, approximate covering is much cheaper than exact covering. Further we present a simple implementation of the algorithm so that our results may directly benefit current implementations of content-based publish/subscribe systems. It is interesting to see if such approximation algorithms for covering also exist for other types of data, such as strings.

Acknowledgments

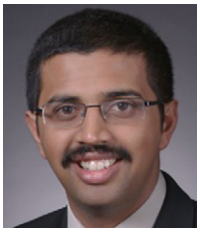
The first author was partly supported through National Science Foundation (NSF) grant 0520102, and the second author was partly supported through NSF grants, 0520102, 0834743, 0831903 and by ICUBE, Iowa State University.

References

[1] A. Carzaniga, M.J. Rutherford, A.L. Wolf, A routing scheme for content-based networking, in: Proc. IEEE INFOCOM 2004, 2004.
[2] Raphaël Chand, Pascal Felber, Xnet: a reliable content-based publish/subscribe system, in: Proc. International Symposium on Reliable Distributed Systems, SRDS, 2004, pp. 264–273.
[3] B. Chazelle, Lower bounds for orthogonal range searching: I. The reporting case, Journal of the ACM 37 (1990) 200–212.
[4] B. Chazelle, Lower bounds for orthogonal range searching: II. The arithmetic model, Journal of the ACM 37 (1990) 439–463.
[5] G. Cugola, E. Di Nitto, A. Fuggetta, The JEDI event-based infrastructure and its application to the development of the OPSS WFMS, IEEE Transactions on Software Engineering 27 (9) (2001) 827–850.
[6] H. Edelsbrunner, M.H. Overmars, On the equivalence of some rectangle problems, Information Processing Letters 14 (3) (1982) 124–127.
[7] C. Faloutsos, Multiattribute hashing using gray codes, in: Proc. ACM SIGMOD Conference on Management of Data, 1986, pp. 227–238.
[8] C. Faloutsos, Gray codes for partial match and range queries, IEEE Transactions on Software Engineering 14 (10) (1988) 1381–1393.
[9] V. Gaede, O. Günther, Multidimensional access methods, ACM Computing Surveys 30 (1998) 170–231.
[10] D. Hilbert, Uber die stetige Abbildung einer Linie auf ein Flachenstuck, Mathematische Annalen 38 (1891) 459–460.
[11] Hojjat Jafarpour, Sharad Mehrotra, Nalini Venkatasubramanian, Mirko Montanari, Mics: an efficient content space representation model for publish/subscribe systems, in: Proc. Third ACM International Conference on Distributed Event-Based Systems, DEBS, 2009, pp. 7:1–7:12.
[12] V.J. Leung, E.M. Arkin, M.A. Bender, D.P. Bunde, J. Johnston, A. Lal, J.S.B. Mitchell, C.A. Phillips, 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, 2002, pp. 296–304.
[13] G. Li, S. Hou, H. Jacobsen, A unified approach to routing, covering and merging in publish/subscribe systems based on modified binary decision diagrams, in: Proc. International Conference on Distributed Computing Systems, ICDCS, 2005, pp. 447–457.
[14] B. Moon, H.V. Jagadish, C. Faloutsos, J.H. Saltz, Analysis of the clustering properties of the Hilbert space-filling curve, The IEEE Transactions on Knowledge and Data Engineering 13 (1) (2001) 124–141.
[15] G.M. Morton, A Computer Oriented Geodetic Data Base and A New Technique in File Sequencing, IBM, Ottawa, Canada, 1966.
[16] G. Mühl, L. Fiege, A.P. Buchmann, Filter similarities in content-based publish/subscribe systems, in: Proc. International Conference on Architecture of Computing Systems, ARCS, 2002, pp. 224–238.
[17] A. Nakano, R.K. Kalia, P. Vashishta, Scalable molecular dynamics visualization, and data management algorithms for materials simulations, IEEE Computing in Science and Engineering 1 (5) (1999) 39–47.
[18] Oracle spatial. 2001. http://www.orafaq.com/faq/spatial.
[19] A. Ouksel, O. Jurca, I. Podnar, K. Aberer, Efficient probabilistic subsumption checking for content-based publish/subscribe systems, in: Proc. ACM/IFIP/USENIX International Conference on Middleware, Middleware, 2006, pp. 121-140.

[20] F.P. Preparata, M.I. Shamos, Computational Geometry: An Introduction, Springer-Verlag, 1985.
[21] Z. Shen, S. Aluru, S. Tirthapura, Indexing for subscription covering in publish–subscribe systems, in: Proceedings of the 18th International Conference on Parallel and Distributed Computing Systems, PDCS, 2005, pp. 328–333.
[22] S. Tirthapura, S. Seal, S. Aluru, A formal analysis of space filling curves for parallel domain decomposition, in: Proc. IEEE International Conference on Parallel Processing, 2006, pp. 505–512.
[23] Duc A. Tran, Thinh Nguyen, A random projection approach to subscription covering detection in publish/subscribe systems, in: CollaborateCom, 2007, pp. 362–369.
[24] P. Triantafillou, A. Economides, Subscription summarization: a new paradigm for efficient publish/subscribe systems, in: Proc. International Conference on Distributed Computing Systems, ICDCS, 2004, pp. 562–571.
[25] M. Warren, J. Salmon, A parallel hashed-octtree N-body algorithm, in: Proc. Supercomputing, 1993, pp. 12–21.
[26] D.E. Willard, New trie data structures which support very fast search operations, Journal of Computer and System Sciences 28 (3) (1984) 379–394.
[27] D.E. Willard, G.S. Lueker, Adding range restriction capability to dynamic data structures, Journal of the ACM 32 (3) (1985) 597–617.
[28] Y. Zhao, D. Sturman, S. Bhola, Subscription propagation in highly-available publish/subscribe middleware, in: Proc. ACM/IFIP/USENIX International Conference on Middleware, Middleware, 2004, pp. 274–293.

Zhenhui Shen received his Ph.D. in Computer Engineering from Iowa State University in 2007, and is currently an Engineer at Akamai Technologies. He is interested in research in efficient content delivery over the Internet.



Srikanta Tirthapura received his Ph.D. in Computer Science from Brown University in 2002, and his B.Tech. in Computer Science and Engineering from IIT Madras in 1996. He is currently an Associate Professor in the department of Electrical and Computer Engineering at Iowa State University. His research is concerned with data-intensive computing and distributed computing.