Exploring Spatial Datasets with Histograms

Chengyu Sun   Divyakant Agrawal   Amr El Abbadi

Department of Computer Science
University of California, Santa Barbara
Email:{cysun, agrawal, amr}@cs.ucsb.edu

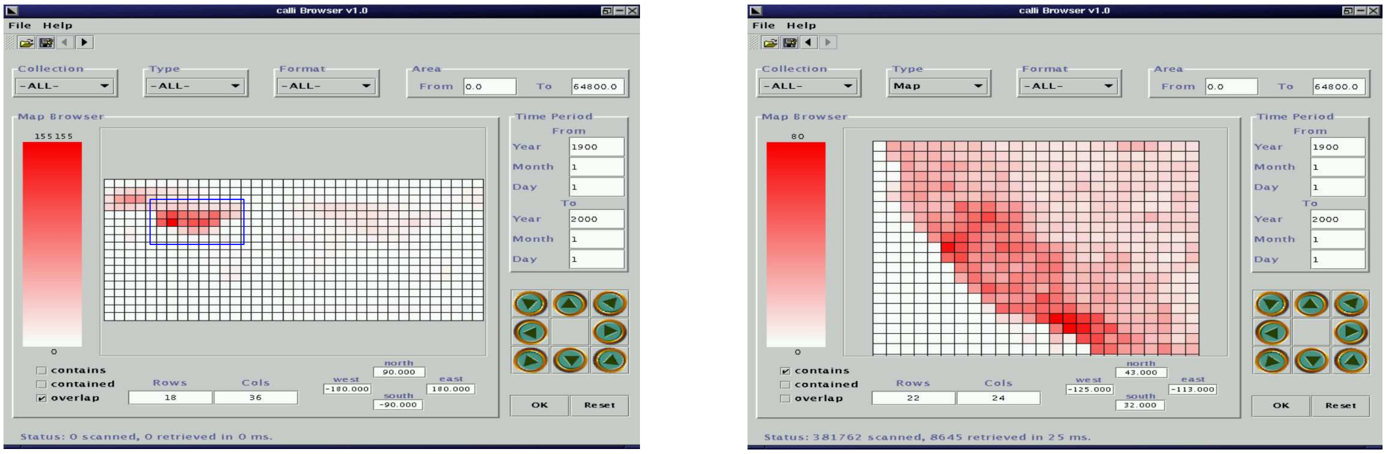
Abstract

As online spatial datasets grow both in number and sophistication, it becomes increasingly difficult for users to decide whether a dataset is suitable for their tasks, especially when they do not have prior knowledge of the dataset. In this paper, we propose *browsing* as an effective and efficient way to explore the content of a spatial dataset. Browsing allows users to view the size of a result set before evaluating the query at the database, thereby avoiding zero-hit/mega-hit queries and saving time and resources. Although the underlying technique supporting browsing is similar to range query aggregation and selectivity estimation, spatial dataset browsing poses some unique challenges. In this paper, we identify a set of spatial relations that need to be supported in browsing applications, namely, the *contains*, *contained* and the *overlap* relations. We prove a lower bound on the storage required to answer queries about the *contains* relation accurately at a given resolution. We then present three storage-efficient approximation algorithms which we believe to be the first to estimate query results about these spatial relations. We evaluate these algorithms with both synthetic and real world datasets and show that they provide highly accurate estimates for datasets with various characteristics.

1 Introduction

Indexing and searching of spatial data have been extensively studied in the last twenty years. A recent survey[1] shows that more than fifty data structures have been developed to support efficient access of objects in spatial/multi-dimensional databases. In contrast, tools to support exploring a whole dataset instead of accessing individual data objects are rather scarce. As online spatial datasets grow both in number and sophistication, finding suitable datasets for a task becomes increasingly difficult. For spatial data archives, Flewelling and Egenhofer [2] identified several ways to evaluate the usefulness of an archive and pointed out that none of them offers an ideal solution. For spatial databases, the user's choices are even more limited, since downloading the complete dataset or part of it is no longer an option. In some cases, the user must rely on the metadata describing the content of the dataset, which is often insufficient and unable to capture the data distribution across multiple attributes. In other cases, the dataset is only accessible through a query interface. Exploring the dataset with trial queries can be a frustrating experience. Due to the user's lack of knowledge of the dataset, trial queries tend to be either overly restrictive or overly broad, resulting in either zero hit or thousands of hits, both of which convey very little information about the dataset itself.

The Alexandria Digital Library (ADL) [3] currently hosts more than 6 million geo-referenced records. One of the goals of the project is to make spatial data more accessible to both researchers and inexperienced users, for example, undergraduate students who take geography classes. One of the problems that arise in using the dataset is that existing metadata do not provide enough information about the characteristics of the dataset. As part of the effort to address this problem, the *GeoBrowsing* service is being developed to provide summary information of a data collection or a subset of it at various resolutions. The browsing service allows users to rapidly gain knowledge of the collection and helps the users to formulate more efficient queries.



(a) Data Distribution Across the World

(b) Map Distribution in California

Figure 1: GeoBrowsing Client Interface

Figure 1 shows the client interface[1] of the GeoBrowsing service. Through this interface, users can make queries based on various data attributes such as region, date and subject type, and the results are shown in the main display area ("Map Browser"), with different colors indicating the number of objects that satisfy the query contraints.

The GeoBrowsing service has two important features to help users to explore a dataset more efficiently. First of all, the system supports *tiles*. When a user selects a region of interest, he or she may choose to further partition the region into *tiles* by specifying the numbers of rows and columns. For instance, Figure 1 (b) shows that California is partitioned into 22×24 tiles. The system interprets each tile, together with the contraints on other attributes, as a single query. So instead of asking the users to send trial queries one by one, this feature essentially allows users to send out hundreds or even thousands of trial queries with a single click, therefore improves the efficiency of exploring a dataset dramatically.

Another important feature of the Geobrowsing service is that it allows users to formulate queries on several spatial relations, including *contains* (the query MBR[2] contains the object MBRs), *contained* (the query MBR is contained in the object MBRs) and *overlap* (the query MBR intersects the object MBRs, but do not form *contains* or *contained* relation). Compared to existing systems [4] which only support the *intersect* spatial

[1]The client interface shown in Figure 1 is a research prototype. We will soon replace it with a new interface provided by ESRI. The new interface supports overlaying the query results on top a map, which greatly improves the usability of the service.

[2]MBR is a minimal bounding rectangle that is used to approximate the spatial extent of a spatial object.

relation (without distinguishing *contains*, *contained* and *overlap* relations), this feature helps users to make more specific queries and therefore get more desirable results.

The current implementation of the GeoBrowsing service prototype builds an index structure on top of the actual data. This implementation meets all the feature requirements and always returns accurate results. However, the performance of the system is not satisfactory when the number of results or the number of tiles is very high. In this paper, we take an alternative approach and try to explore the tradeoff between speed and accuracy. We further simplify the problem by considering only the spatial attribute, and focus our attention on supporting different spatial relations.

The rest of the paper is organized as follows. In Section 2, we discuss several issues related to the browsing service, in particular, the spatial relations that need to be supported. In Section 3 we introduce two existing algorithms, both of which are very efficient and can give exact results for queries about the *intersect* spatial relation at a given resolution. However, for more specific spatial relations such as *contains*, we show that the storage requirements of similar algorithms will be prohibitively high even at a moderate resolution. So instead of trying to give exact results, we develop three efficient approximation algorithms. Section 4 introduces the theoretical basis of the algorithms, and Section 5 discusses the algorithms in details. We evaluate these algorithms with both synthetic and real datasets, and analyze the results in Section 6. Section 7 concludes the paper and discusses some future work.

2 Spatial Database Browsing and Related Issues

From a functional perspective, a browsing system can be considered as a database that can process a group of queries simultaneously. So when users try to explore a dataset, instead of sending out individual trial queries, they can simply select the whole dataset, grid it into *tiles*, and send out the queries for all the tiles with a single command.

There are certain aspects of a browsing system that have been studied in prior research. The Human-Computer Interaction Laboratory at University of Maryland at College Park (HCIL-UMD) has been working on similar systems [4, 5] since 1996 with an emphasis on user interfaces. The query processing part of a browsing system, which returns the size of a result set rather than the actual objects, is closely related to the work in the areas of range query aggregation and range query selectivity estimation. However, spatial database browsing raises some unique issues that have not been addressed before.

Since spatial objects typically span a range in space, a spatial database browsing system should be able to handle not only point objects, but also *range objects* such as line segments, rectangles and polygons. Among these object types, rectangular objects are particularly important because different types of objects can be represented by their Minimal Bounding Rectangles (MBRs). Although range query aggregation has been extensively studied and a number of data structures such as data cubes have been proposed [6, 7, 8, 9], to the best of our knowledge, no existing data structures are designed explicitly for rectangular objects. One might argue that 2-dimensional rectangles can be regarded as 4-dimensional points, therefore can be handled by existing data structures. While there is much truth in this statement, it is often undesirable to treat rectangles as points in practice due to performance reasons. Currently, the most query-efficient aggregation technique is the prefix-sum data cube [7], which achieves constant query response time. The disadvantage of the prefix-sum data cube is that the number of cells in the data cube increases exponentially as the dimensionality increases. For example, if we partition the earth into $1^\circ \times 1^\circ$ regions, only $360 \times 180 = 64,800$ cells are needed; but if we treat the rectangles as 4-d points, four billion ($360 \times 180 \times 360 \times 180$) cells are needed. There are other data cube structures that are more storage-efficient. For instance, the number of cells in pCube [10] is proportional to the size of the original dataset. But the storage efficiency comes at the cost of query efficiency, which makes this types of data structures not applicable for browsing applications. Note that in spatial database browsing, a browsing query typically consists of a 2-dimensional array of *tiles*, and each tile can be considered as a COUNT aggregation query by itself. The number of tiles could easily reach hundreds or even thousands. For example, partitioning the earth at a moderate resolution of $10^\circ \times 10^\circ$ yields 648 tiles. The sheer number of tiles demands very efficient algorithms for the browsing service.

Another area of research that is closely related to the browsing applications considered in this paper is spatial range query selectivity estimation [11, 12, 13]. We will discuss one of these algorithms, the Cumulative Density algorithm [13], together with Beigel-Tanin's algorithm [5] in more details in Section 3. These algorithms are designed to handle range objects, and are usually very efficient in both storage space and query response time. However, these algorithms only distinguish between two types of spatial relations: *disjoint* and *intersect*, while spatial database users are often interested in a much richer set of spatial relations.

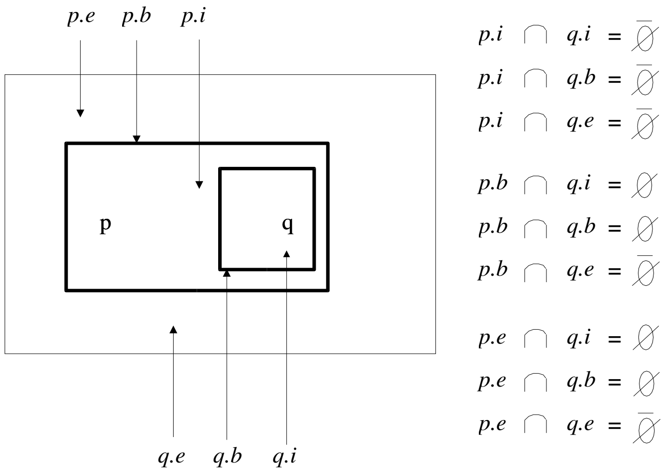


Figure 2: Results of the 9 Intersections When p *contains* q

One of the spatial relation models, the *9-intersection model* [14], defines the spatial relationship between two region objects based on the 9 intersections of their interiors, exteriors and boundaries. Formally, let p and q be two region objects, and $p.i$, $q.i$, $p.b$, $q.b$, $p.e$ and $q.e$ be their interiors, boundaries and exteriors, then the spatial relationship between p and q can be defined by the following 3×3 matrix:

$$\begin{bmatrix} p.i \cap q.i & p.i \cap q.b & p.i \cap q.e \\ p.b \cap q.i & p.b \cap q.b & p.b \cap q.e \\ p.e \cap q.i & p.e \cap q.b & p.e \cap q.e \end{bmatrix} \tag{1}$$

Figure 2 shows the results of the nine intersections when object p *contains* object q.

Although mathematically the 9-intersection model can distinguish 2^9 spatial relations, not all of them are physically possible. For example, no matter how two objects are positioned, their exteriors always intersect, so $p.e \cap q.e$ is always 1. For two region objects without holes, it is proved in [14] that only eight spatial relations exist. These eight spatial relations and their corresponding intersection matrices are shown at the bottom of Figure 3, and we will call this set of relations the Level 3 Spatial Relations.

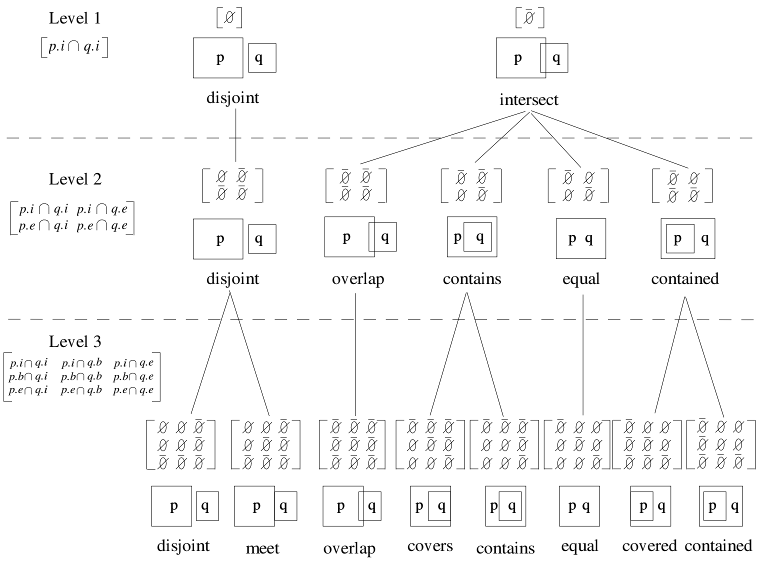


Figure 3: Spatial Relations at Different Levels

Note that the *disjoint* and *intersect* spatial relations supported by range query selectivity estimation techniques can be defined by using only the intersection of the interiors of two objects, as shown at the top of Figure 3. We call this set of spatial relations the Level 1 Spatial Relations. Existing techniques [11, 12, 13, 5] can handle queries about Level 1 spatial relations very efficiently using auxiliary data structures such as histograms. However, to answer queries about more specific spatial relations such as *contains*, the actual data objects must be accessed. Our experience with the GeoBrowser prototype shows that this approach is not efficient for browsing spatial datasets.

In this paper, we try to bridge the gap between Level 3 and Level 1 Spatial Relations by introducing a new spatial relation model, called the *interior-exterior intersection model*[3], which is derived from the 9-intersection model by removing the intersections involving the object boundaries. Under this model, the spatial relation between two objects can be defined by the following 2×2 matrix:

$$\begin{bmatrix} p.i \cap q.i & p.i \cap q.e \\ p.e \cap q.i & p.e \cap q.e \end{bmatrix} \tag{2}$$

For region objects without holes, the interior-exterior intersection model can distinguish five spatial relations, which we call the Level 2 Spatial Relations (shown in Figure 3). The rationale behind the interior-exterior intersection model is that for spatial database browsing, the boundary relations such as *meet* and *covers* are not important. Users of a browsing service are likely to be interested in learning the dataset content at

[3] The term *4-intersection model* is already taken [14]. The 4-intersection model defines spatial relations based on the intersections of object interiors and boundaries.

a high level, so omitting certain details does not decrease the value of the service. More importantly, the interior-exterior model enables efficient implementations that do not rely on accessing the actual data objects, as shown later in Section 5.

In the rest of the paper, all spatial relations such as *overlap*, *contains* and *contained* refer to the Level 2 spatial relations unless specified otherwise. The only exception is *intersect*, which is a Level 1 spatial relation and should not cause any confusions. Also note that all spatial relations discussed in the rest of the paper are with respect to a query (which corresponds to object p in Figure 3). For example, the number of objects that satisfy the *contains* spatial relation are the number of objects that *are contained* in the query MBR.

3 Exact Algorithms for Spatial Relations

For spatial database browsing, we are particularly interested in techniques that can return exact aggregation results *at a given resolution*. The rationale is that details are not important in exploring a dataset. At a certain resolution, users will have enough information to decide whether the dataset is useful or not. If the dataset is found out to be useful, users can then proceed to send queries to the database and access the actual data.

Formally, let S be a set of d-dimensional objects and R^d be a hyper-rectangle that encloses all the objects in S. A *gridding* of R^d partitions each dimension D_i of R^d into n_i equi-width segments, so R^d is partitioned into $\prod n_i = N$ equi-sized *cells*. We use a unit cell c to represent the *resolution* of the grid. If the MBR of query Q completely aligns with the grid, we say the query is at resolution c; and if an algorithm can return exact aggregation results for all queries about spatial relation r at resolution c, we say this algorithm is an *exact algorithm for spatial relation* r.

Note that for point data, an exact algorithm for the *contains* spatial relation can be easily developed by using a histogram with each histogram bucket corresponding to a grid cell. However, the same approach does not apply to rectangular objects which may span several cells. For example, in the Minskew algorithm [11], if an object spans several histogram buckets, it is counted once in each bucket. So for a query covering several histogram buckets, the result may not be accurate because one object could be counted multiple times. In this section, we first introduce two existing techniques that address this problem.

3.1 Algorithms for the *intersect* Spatial Relation

The Cumulative Density (CD) algorithm is one of the techniques proposed in [13] to approximate range query selectivity in spatial databases. Assuming all objects are represented by their MBRs, the CD algorithm can be adapted for browsing applications as follows: given a grid of R^2 at resolution c, construct four histograms H_{ll}, H_{lr}, H_{ul} and H_{ur}. The size of each histogram is N with each bucket corresponding to a grid cell. A bucket of H_{ll} keeps the number of lower-left vertices that fall into the bucket. Similarly, H_{lr}, H_{ul} and H_{ur} keep the counts of lower-right, upper-left and upper-right vertices of the objects. To improve query efficiency, all the histograms are *cumulative*, in the sense that a bucket $H(i, j)$ stores the number of vertices in the region $(0, 0, i, j)$. So for a query (x_a, y_a, x_b, y_b), the number of intersecting objects can be calculated as follows:

$$N_{intersect} = H_{ll}(x_b, y_b) - H_{lr}(x_a - 1, y_b) - H_{ul}(x_b, y_a - 1) + H_{ur}(x_a - 1, y_a - 1) \tag{3}$$

Figure 4 (a) shows an example of three objects and a query at (x_a, x_b, y_a, y_b). Figure 4 (b) are the four histograms constructed by the CD algorithm. The number of objects that intersect the query is $H_{ll}(x_b, y_b) - H_{lr}(x_a - 1, y_b) - H_{ul}(x_b, y_a - 1) + H_{ur}(x_a - 1, y_a - 1) = 3 - 0 - 1 + 0 = 2$.

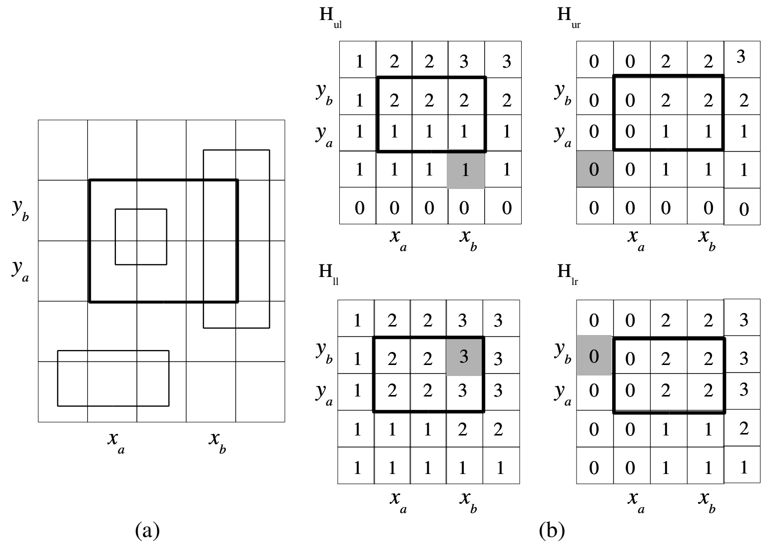


Figure 4: The Cumulative Density (CD) Algorithm

For any queries at resolution c, the CD algorithm returns the exact number of objects that *intersect* the query. The storage requirement is $2^d N$ for a grid of R^d, and it takes 2^d lookups to answer a query so the query response time is constant.

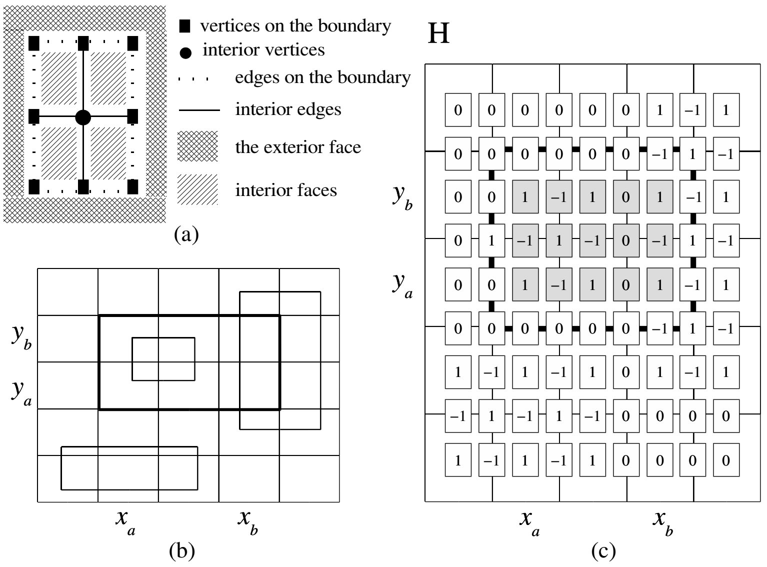


Figure 5: Beigel-Tanin's (BT) Algorithm

Another exact algorithm for the *intersect* spatial relation was proposed by Beigel and Tanin in [5]. We refer to this algorithm as the Beigel-Tanin's (BT) algorithm. For a grid of R^2, the BT algorithm constructs a histogram H with each bucket corresponding to an *interior* vertex, edge or face of the grid. A vertex or an edge is *interior* if it is not on the boundary of the grid, and a face is interior if it is not the *exterior face* (the "space" outside the grid). As shown in Figure 5 (a), a 2×2 grid has one interior vertex, four interior edges and four interior faces.

The value held in a bucket of the histogram H is determined as follows: if the bucket corresponds to an interior vertex or face, it stores the count of the objects that intersect the vertex or face; if a bucket corresponds to an edge, it stores the *negative value* of the intersecting object count. Figure 5 (b) shows an example of three objects, and Figure 5 (c) shows the corresponding histogram H. With the histogram H, the number of objects that intersect a query Q can be calculated as

$$N_{intersect} = \sum H(b_i) \tag{4}$$

where b_i is a bucket whose corresponding vertex, edge or face intersects the interior of Q, and $H(b_i)$ is the value stored in b_i. As the example shown in Figure 5 (b) and (c), the number of objects that intersect the query at (x_a, x_b, y_a, y_b) is the sum of all the shaded buckets in Figure 5 (c), which is $1 + (-1) + 1 + 0 + 1 + (-1) + 1 + (-1) + 0 + (-1) + 1 + (-1) + 1 + 0 + 1 = 2$.

As with histograms in the CD algorithm, histogram H can be cumulative, so the time to answer a rectangular-shaped query is constant. For a grid of R^d, the storage requirement of H is $\prod_{0 \leq i \leq d}(2n_i - 1) = O(N)$.

3.2 Algorithms for Level 2 Spatial Relations

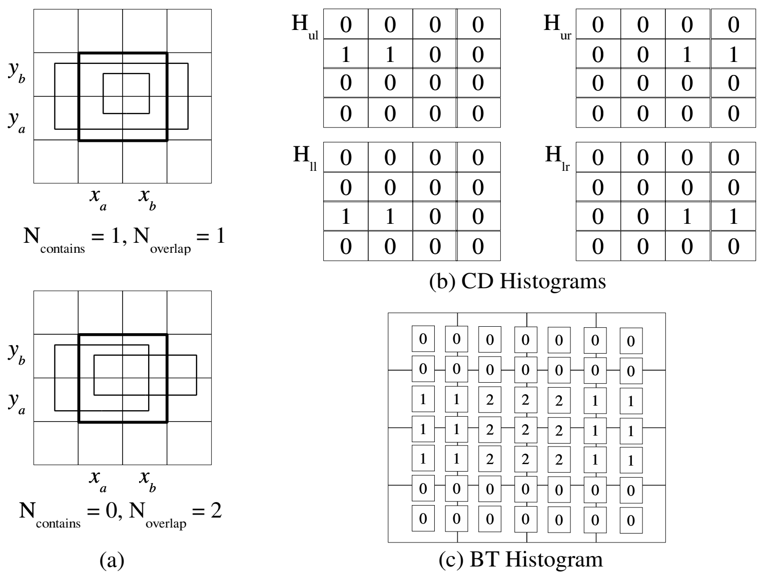


Figure 6: CD and BT Algorithms Cannot Handle Level 2 Spatial Relations

Both the CD and BT algorithms are efficient algorithms which return exact results for the *intersect* spatial relation at a given resolution, but neither of them can handle Level 2 spatial relations such as *contains*. Figure 6 (a) shows two different scenarios. For a query at (x_a, x_b, y_a, y_b), the *contains* result should be 1 in the first case and 0 in the second case. Note that both the CD and BT algorithms construct identical histograms for these two scenarios (shown in Figure 6 (b) and (c) where we omit the cumulative step), which means that based on the information kept in these histograms, one can not tell the exact number of objects that are contained in the query rectangle.

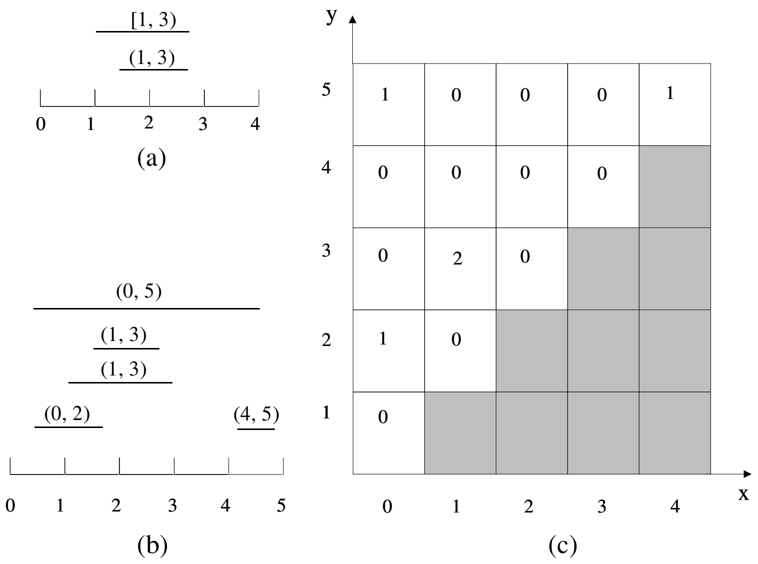


Figure 7: A 2-D Histogram for 1-D Range Data

To find out how much information is required to identify the Level 2 spatial relations, let us first analyze the 1-dimensional case. For a dataset S of 1-dimensional range objects and a grid of R^1, we first make a simplification and assume that no objects align with the grid. The reason for this simplification is that an object that starts *from* i and ends before j (which can be denoted as $[i, j)$) is different from an object that starts *after* i and ends before j (which can be denoted as (i, j)). Figure 7 (a) shows such an example. Note that object $[1, 3)$ contains the range $[1, 2]$ while object $(1, 3)$ overlaps the range $[1, 2]$.

With this simplification, all objects in S can be represented by (i, j) and $0 \leq i < j \leq n$, so we can construct a 2-dimensional histogram H with each bucket h_{ij} containing the number of objects (i, j), as illustrated in Figure 7 (b) and (c). Since $i < j$, the effective size of the histogram is $|H| = n(n + 1)/2 = O(N^2)$.

Histogram H has two important properties:

- Any query at the given resolution can be answered *exactly* with H.
- The values in each bucket are *independent* of each other. For example, the number of objects between 1 and 2 has nothing to do with the number of objects between 1 and 3.

Assume there exists an algorithm which can return exact results for the *contains* relation for *any* datasets by keeping a *constant* number of values $V = v_1, v_2, ..., v_n$, and $|V| < |H|$. Since H contains the complete information of a dataset at a given resolution, there must exist a mapping from the values stored in H to the values in V, and this mapping can be expressed as an equation array with $|V|$ equations and $|H|$ variables. Now let $contains(i, j)$ be the number of objects that are contained in the range $[i, j]$, then we can use this algorithm to compute H as follows:

$$H(m, n) = contains(m, n) - \sum_{m \leq i < n} \sum_{i < j \leq n} contains(i, j) \tag{5}$$

Since $contains(i,j)$ can be obtained from V, it follows that we can compute H from V, which contradicts the fact that at least $|H|$ equations are needed to solve $|H|$ independent variables. We can also see this from another perspective: $|H|$ values cannot be losslessly compressed to $|V|$ values unless the original values contain redundent information. Since the values in H are independent of each other, for any algorithm we can always construct a dataset that requires the algorithm to keep at least $|H|$ values.

We now prove the following theorem for d-dimensional cases:

Theorem 3.1 *Let S be an arbitrary set of d-dimensional rectangular objects and R^d be a hyper-rectangle that encloses all objects in S. Given a $n_1 \times n_2 ... \times n_d$ grid of R^d, an algorithm that can return exact results for the contains spatial relation requires at least $\prod_{1 \leq i \leq d} n_i(n_i+1)/2 = O(N^2)$ storage space, where $N = n_1 \times n_2 \times ... \times n_d$.*

Proof. As in the 1-d dimensional case, we first construct H, which is a $2d$-dimensional histogram. The effective size of H can be calculated as follows:

Let $x_1, x'_1, x_2, x'_2, ..., x_d, x'_d$ be the $2d$ dimensions of H, and $h_{a_1 a'_1 a_2 a'_2 ... a_d a'_d}$ be a bucket of H. Note that the $2d$ dimensions of H can be considered as d pairs of dimensions (x_i, x'_i), while x_i is the dimension for the start points of the objects in the original ith dimension, and x'_i is the dimension for the end points. In order for $h_{a_1 a'_1 a_2 a'_2 ... a_d a'_d}$ to be valid, a'_i must be greater than a_i, which means that for each dimension pair (x_i, x'_i), there are only $n_i(n_i+1)/2$ valid combinations of (a_i, a'_i). So the effective size of H is $|H| = \prod_{1 \leq i \leq d} n_i(n_i+1)/2$.

The next step is to show that we can construct H with an algorithm that returns exact results for the *contains* spatial relation. Let $A = (a_1, a'_1, ..., a_d, a'_d)$ be a d-dimensional range, then $h_{a_1 a'_1 a_2 a'_2 ... a_d a'_d}$ is the number of objects contained in A subtracting the number of objects contained in the d-dimensional subranges of A, so

$$h_{a_1 a'_1 ... a_d a'_d} = contains(a_1, a'_1, ... a_d, a'_d) - \sum_{a_1 \leq b_1 < n_1} \sum_{b_1 < b'_1 \leq n_1} ... \sum_{a_d \leq b_d < n_d} \sum_{b_d < b'_d \leq n_d} contains(b_1, b'_1, ..., b_d, b'_d) \quad (6)$$

Since H contains $\prod_{1 \leq i \leq d} n_i(n_i+1)/2$ independent variable, an algorithm must keep at least $\prod_{1 \leq i \leq d} n_i(n_i+1)/2$ values in order to return exact results for *contains*.

□

We make the following observations about Theorem 3.1:

- The correctness of Theorem 3.1 is based on Equation 5 (or Equation 6 for d-dimensional cases), which states that if we know the *contains* results for all possible ranges $[i,j]$, we will be able to compute all values in H. However, the equivalent of Equation 5 does not exist for the *intersect* relation, which means that exact algorithms for *intersect* requires less than $O(N^2)$ space. In fact, as demonstrated by the CD and BT algorithms, such algorithms only require $O(N)$ space.

- Theorem 3.1 indicates that exact algorithms for the Level 2 spatial relations are very expensive in terms of storage requirement. In fact, it is often infeasible for even 2-dimensional cases. For example, given a $360° \times 180°$ space and a grid at the resolution of $1° \times 1°$, such an algorithm requires $4 \times (360 \times 361)/2 \times (180 \times 181)/2 \simeq 4GB$ space.

- Given a data space and the grid resolution, Theorem 3.1 gives the storage lower bound for an algorithm to answer *contains* query accurately for *any* dataset. This means that for a particular dataset (for example, a very small dataset), it would be more storage-efficient to simply use the original dataset to answer *contains* queries. However, as we discussed in Section 2, the performance of this approach will not be good for browsing large datasets.

- In the previous discussions, we have assumed that no object aligns with the grid, so all objects are of the type (i, j). We can extend this model to more general cases and include objects of the types $[i, j)$, $(i, j]$ and $[i, j]$. This will increase the storage requirement by a constant factor of 4.

4 Theoretical Background

Since Theorem 3.1 indicates that an exact algorithm for the Level 2 spatial relations is likely to be either space-expensive or time-expensive for large datasets, we would like to develop an efficient approximation algorithms that can give reasonably accurate results for the Level 2 spatial relations. The algorithms we propose are based on Euler's Formula and the interior-exterior model we introduced in Section 2.

4.1 Euler's Formula and Corollaries

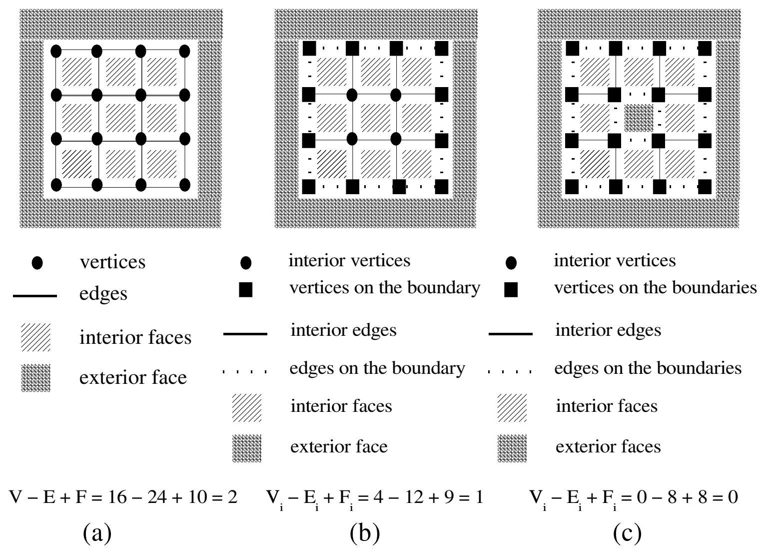


Figure 8: Euler's Formula and Corollaries

Euler's Formula [15] is an important result in graph theory, which states:

Theorem 4.1 *For any graph with V vertices, E edges and F faces,*

$$V - E + F = 2 \tag{7}$$

Figure 8 (a) shows an example of Euler's Formula where the graph is a 3×3 grid. Note that the *exterior* of a graph is also counted as a face, so in the example, the number of faces is 10 instead of 9.

In [5], Beigel and Tanin proved a corollary of the Euler's Formula. For the purpose of this paper, we only present the 2-dimensional version of the corollary here:

Corollary 4.1 *Let S be a bounded graph. A vertex, edge or face of S is an interior vertex, edge or face if it is not the exterior face and it is not entirely contained in the boundary of S. Let V_i, E_i and F_i be the number of interior vertices, interior edges and interior faces, then*

$$V_i - E_i + F_i = 1 \tag{8}$$

An example illustrating Corollary 4.1 is shown in Figure 8 (b) where we use the same 3×3 grid. After removing the exterior face and the boundary, we now have 4 interior vertices, 12 interior edges and 9 interior faces, and have the formula evaluate to 1.

We now extend Beigel and Tanin's corollary to handle graphs that have more than one exterior faces. Figure 8 (c) shows a graph with two exterior faces, corresponding to a region with a "hole", which we will encounter later when we discuss the approximation algorithms. Comparing Figure 8 (b) and (c), we can see that the face in the middle is now an exterior face. Consequently, the vertices and edges around this exterior face are no longer interior vertices or edges. After removing the two exterior faces and the two boundaries, we have 0 interior vertices, 8 interior edges and 8 interior faces, so $V_i - E_i + F_i = 0$.

For graphs with k exterior faces, we give the following corollary with a brief proof. A more rigorous proof can be done using the Euler characteristics in algebraic topology. Interested readers are refered to [16] for more details.

Corollary 4.2 *Let S be a bounded graph with k exterior faces and no two exterior faces share the same boundary. A vertex, edge or face of S is an interior vertex, edge or face if it is not one of the exterior faces and it is not entirely contained in one of the boundaries of S. Let V_i, E_i and F_i be the number of interior vertices, edges and faces, then*

$$V_i - E_i + F_i = 2 - k \tag{9}$$

Proof. Let V_{b_i} and E_{b_i} be the number of vertices and edges on a boundary b_i. From Euler's Formula, for a graph with k exterior faces, we have

$$V - E + F = V_i + \sum_{0 \leq i \leq k} V_{b_i} - E_i - \sum_{0 \leq i \leq k} E_{b_i} + F_i + k = 2 \tag{10}$$

Note that a boundary can be considered as a graph with two faces: an interior face and an exterior face. Again from Euler's Formula, we have $V_{b_i} - E_{b_i} + 2 = 2$, or $V_{b_i} - E_{b_i} = 0$. Substituting this result into Equation 10, we have $V_i - E_i + F_i = 2 - k$. □

4.2 Basic Equations from the Interior-Exterior Intersection Model

Let Q be a query and S a dataset. Under the interior-exterior model introduced in Section 2, there are only five possible spatial relations between Q and an object in S. Let N_d, N_{cs}, N_{cd}, N_{eq} and N_o to be the number of objects in S that satisfy the five Level 2 spatial relations *disjoint*, *contains*, *contained*, *equals* and *overlaps* with respect to Q. Let

- n_{ii} to be the number of objects whose interiors intersect the interior of Q
- n_{ie} to be the number of objects whose exteriors intersect the interior of Q
- n_{ei} to be the number of objects whose interiors intersect the exterior of Q
- n_{ee} to be the number of objects whose exteriors intersect the exterior of Q

From the definition of the Level 2 spatial relations (Figure 3), we have the following equation:

$$N_d \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} + N_{cs} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} + N_{cd} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} + N_{eq} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + N_o \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} n_{ii} & n_{ie} \\ n_{ei} & n_{ee} \end{bmatrix} \tag{11}$$

Note that $n_{ee} = N_d + N_{cs} + N_{cd} + N_{eq} + N_o$. Since $(N_d + N_{cs} + N_{cd} + N_{eq} + N_o)$ is the total number of objects in the dataset S, and the size of S is usually a known value, we can replace n_{ee} with $|S|$ in Equation 11, and get

$$N_d \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} + N_{cs} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} + N_{cd} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} + N_{eq} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} + N_o \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} n_{ii} & n_{ie} \\ n_{ei} & |S| \end{bmatrix} \tag{12}$$

As discussed in Section 2, spatial relations that involve boundaries such as *equals* are not important in the browsing applications, so we would like to eliminate N_{eq} from Equation 12. In practice, this can be done by "shrinking" an object a little bit if its boundary completely aligns with a given grid. For instance, for a 1-d object $[1, 3]$, we treat it as if it is $(1, 3)$. So for all queries at the given resolution, the result of N_{eq} is always 0, and Equation 12 becomes:

$$N_d \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} + N_{cs} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} + N_{cd} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} + N_o \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} n_{ii} & n_{ie} \\ n_{ei} & |S| \end{bmatrix} \tag{13}$$

Clearly, for a query Q, if we can find out the values of n_{ii}, n_{ei} and n_{ie}, we would be able to find out N_d, N_{cs}, N_{cd} and N_o by solving Equation 13. In fact, Equation 13 can be simplified even further. Note that many real world datasets contain primarily small[4] objects. Also for non-point queries, a user usually specifies a reasonable-sized query which will not be completely contained in any objects. So in many cases, it is reasonable to assume that the results of the *contained* queries, or N_{cd}, are always 0, so Equation 13 is reduced to:

$$N_d \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} + N_{cs} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + N_o \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} n_{ii} \\ n_{ei} \\ |S| \end{bmatrix} \tag{14}$$

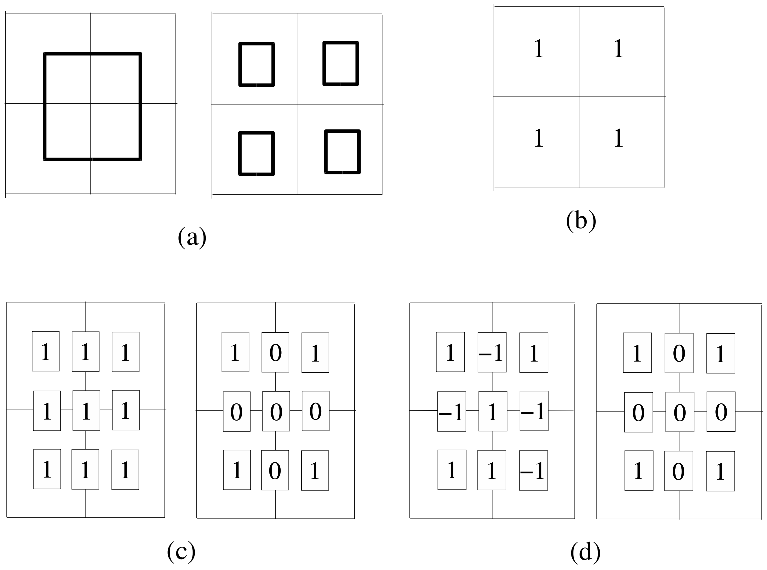


Figure 9: Two Approaches to Construct a Histogram

5 Approximation Algorithms

5.1 A Histogram-based Approach

Given a grid of the data space, a straightforward way to construct a histogram is to let each histogram bucket correspond to a grid cell, and if an object intersects a cell, increment the value of the corresponding bucket by 1. However, as shown in Figure 9 (a) and (b), such a histogram cannot distinguish the difference between one big object that spans several cells and several small objects that are each contained in an individual cell. Comparing the two cases in Figure 9 (a), we can see that the main difference between the two cases is that the big object crosses the cell boundaries while the small objects do not. Based on this observation, we can construct a histogram which not only keeps information for each cell, but also keeps information on the edges and vertices. More precisely, we can construct a histogram H as follows:

- Given a $n_1 \times n_2$ grid of R^2, allocate $(2n_1 - 1)(2n_2 - 1)$ buckets for the histogram H. A bucket of H corresponds to a vertex, an edge or a cell of the grid.
- Scan through the dataset. For each object, if a vertex, an edge or a cell of the grid intersects the interior of the object, increment the corresponding bucket by 1. Figure 9 (c) shows two examples of H after this step. Note that the two different cases in Figure 9 (a) now result in two different histograms.
- Once the whole dataset is processed, invert the values in those buckets that correspond to edges, as shown in Figure 9 (d). The reason for this step is related to the properties of Euler's Formula and will become clear in the next three paragraphs.

This histogram is the same as the one proposed in the BT algorithm [5]. In fact, our algorithms are partially inspired by the BT algorithm. The difference is that we try to use this histogram to answer queries about the Level 2 spatial relations, which is significantly more complex than handling the Level 1 *intersect* relation.

[4] *small* with respect to the size of a grid cell.

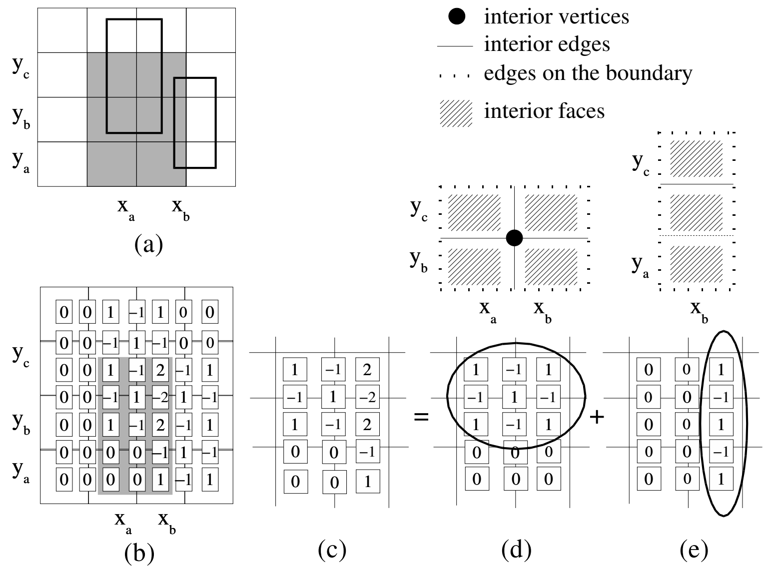


Figure 10: Compute n_{ii}

To use the histogram H to compute n_{ii}, which is the number of objects that *intersect* a query, let us first consider a simple example with two objects and a query at (x_a, x_b, y_a, y_c), as shown in Figure 10 (a) (the shaded region is the query region). Both of these objects intersect the query, and result in two intersecting regions (x_a, x_b, y_b, y_c) and (x_b, x_b, y_a, y_c).

Figure 10 (b) shows the histogram H that corresponds to the dataset in Figure 10 (a). Now consider the buckets of H that are inside the query region (excluding the query boundary), as shown in Figure 10 (c). Note that these buckets can be decomposed into two components. The first component (Figure 10(d)) consists of 9 non-zero buckets, which correspond to the 1 interior vertex ($V_i = 1$), 4 interior edges ($E_i = 4$) and 4 interior faces ($F_i = 4$) of the first intersecting region (x_a, x_b, y_b, y_c). From Corollary 8 of Euler's Formula, $V_i + (-E_i) + F_i = 1$, we know that the sum of these 9 buckets is 1. Similarly, the 5 non-zero buckets of the second component correspond to the interior edges and faces of the second intersecting region (x_b, x_b, y_a, y_c), and the sum of these 5 buckets is also 1. Based on this observation, we can conclude that to calculate the number of objects that intersect a query, or $N_{intersect}$, we can simply sum up all the buckets of H that are inside the query region, because *each intersecting region contributes 1 to the sum.*

Since $n_{ii} = N_o + N_{cd} + N_{cs} = N_{intersect}$ (Equation 11), we have

$$n_{ii} = \sum_b H(b) \tag{15}$$

where b is a bucket that inside the query region.

5.2 Simple Euler Approximation Algorithm

In this subsection we introduce an approximation algorithm which we called Simple Euler Approximation (S-EulerApprox). This algorithm is based on Equation 14, which assumes that the number of objects that contain the query, or N_{cd}, is always 0. For convenience, we reproduce Equation 14 here:

$$N_d \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} + N_{cs} \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + N_o \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} n_{ii} \\ n_{ei} \\ |S| \end{bmatrix}$$

Since the dataset size $|S|$ is usually known, and n_{ii} can be computed as described in Section 5.1, now we only need to compute the value of n_{ei}. Intuitively, since the histogram H keeps the information about the interiors of the objects, and n_{ii} can be computed by summing up all the buckets *inside* the query rectangle, then n_{ei}, which is the number of objects whose interiors intersect the exterior of the query, can be computed by summing up all the buckets that are *outside* the query rectangle (excluding the query boundary). So

$$n_{ei} = \sum_{b_e} H(b_e) \tag{16}$$

where b_e is a bucket that is outside the query region.

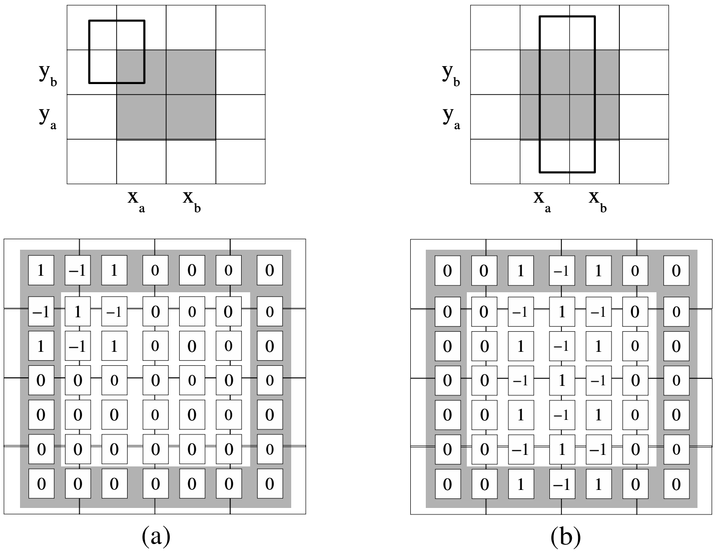


Figure 11: Compute n_{ei}

Figure 11 (a) illustrates an example for computing n_{ei}. The query is at (x_a, x_b, y_a, y_b), and to compute n_{ei}, we simply sum up all the buckets that are outside the query rectangle (the shaded buckets of the histogram in Figure 11 (a)). In this case, we have $1 + (-1) + 1 + (-1) + 1 = 1$, which is the correct result.

However, unlike n_{ii}, the value of n_{ei} computed from Equation 16 is not alway accurate due to *crossover objects*. Informally, a *crossover object* is an object that "crosses" the query rectangle, as the one shown in Figure 11 (b). When the interior of a crossover object intersects the exterior of a query, the result is *two* intersecting regions. Consider the case in Figure 11 (b) as an example: if we sum up all the buckets that are outside the query rectangle, we have $1 + (-1) + 1 + 1 + (-1) + 1 = 2$, while the correct result is 1. We expect the number of crossover objects to be generally small unless the query rectangle is long and narrow, which is very unlikely in the browsing applications, or, an object is long and narrow, which is rare in geo-spatial datasets.

The S-EulerApprox algorithm can be summarized by the following equations:

$$
\begin{aligned}
n_{ii} &= \sum_{b_i} H(b_i) && (17)\\
n_{ei} &= \sum_{b_e} H(b_e) && (18)\\
N_{cs} &= |S| - n_{ei} && (19)\\
N_o &= n_{ei} - N_d = n_{ei} - (|S| - n_{ii}) && (20)
\end{aligned}
$$

where b_i is a bucket inside the query rectangle and b_e is a bucket outside the query rectangle.

5.3 Euler Approximation Algorithm

When the number of objects that contain the query rectangle, or N_{cd}, is comparable to N_{cs} or N_o either because the dataset has a large number of big objects or because the query rectangle is sufficiently small, the assumption of the S-EulerApprox algorithm is no longer valid. In this case, to answer queries about Level 2 spatial relations, we need a more sophisticated algorithm, which we call the Euler Approximation algorithm (EulerApprox). This algorithm is based on Equation 13. Again for convenience, we reproduce Equation 13 here:

$$N_d \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} + N_{cs} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} + N_{cd} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} + N_o \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} n_{ii} & n_{ie} \\ n_{ei} & |S| \end{bmatrix}$$

Since $|S|$ is usually known and n_{ii} can be computed as described in Section 5.1, our task is to compute the values of n_{ei} and n_{ie}. First we need to revisit the computation of n_{ei}, since $n_{ei} = N_d + N_{cd} + N_o$, and we can no longer assume N_{cd} is 0.

If an object contains the query rectangle, then the interior of the object intersects the exterior of a query. If we compute n_{ei} by adding up all the buckets of H that outside of the query rectangle as discussed in Section 5.2, one would expect that the value of n_{ei} will include N_{cd}. Unfortunately, this is not the case, as shown by the example in Figure 12.

Figure 12 shows a query at (x_b, x_c, y_b, y_c) and an object that contains the query. The corresponding histogram is shown in Figure 12 (c). Note that the intersecting region of the object interior and the query exterior is the region (x_a, x_d, y_a, y_d) with a "hole" at (x_b, x_c, y_b, y_c). From Corollary 4.2, we know that for this type of regions, $V_i - E_i + F_i = 2 - k = 0$ (because a hole is an exterior face, the number of exterior faces $k = 2$). In other words, this intersecting region does not contribute to n_{ei}, which can be verified by adding up all the shaded buckets in Figure 12 (c). We call this effect the *loophole* effect. Due to the loophole effect, adding up the buckets outside the query rectangle does not give the correct result of n_{ei} as specified in Equation 13, so we use a different notation n'_{ei}, and let $n'_{ei} = \sum_{b_e} H(b_e)$, where n'_{ei} ignores all objects containing the query.

Compared to n_{ei}, computing the value of n_{ie} is an even bigger challenge. Intuitively, since n_{ie} is the number of objects whose exteriors intersect the query interior, we can construct a histogram H_e in a similar way as we constructed the histogram H, except that histogram H_e keeps the information about object exteriors as opposed to the information about object interiors kept in H. With H_e, we might be able to compute the value of n_{ie} by adding up all the buckets of H_e that are inside the query rectangle. Due to space constraints, we omit a detailed analysis of H_e, but it is sufficient to say that this approach also suffers from the *loophole*

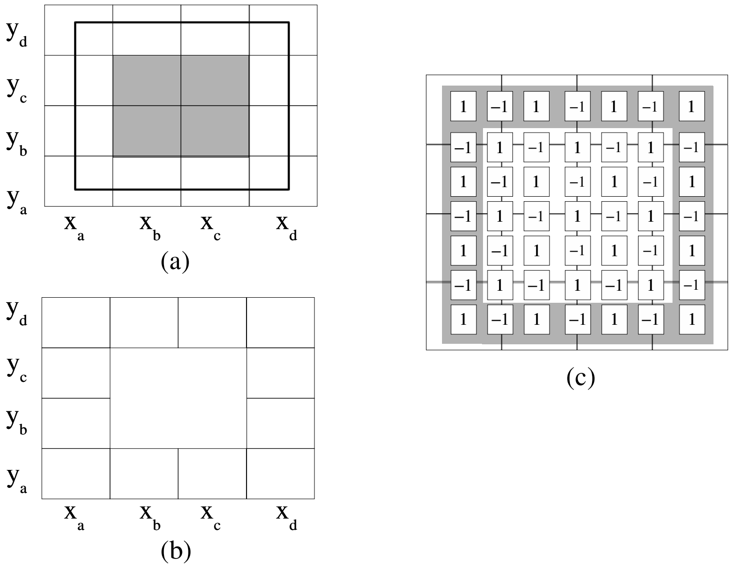


Figure 12: The *loophole* Effect

effect. A histogram H_e does provide some additional information about the dataset, but it does not help unless the query is of the same size as a unit cell of the grid.

Although computing n_{ie} is very difficult, and we only have n'_{ei} instead of n_{ei}, it is still possible to develop a good approximation algorithm with only histogram H. Note that we have four variables N_d, N_o, N_{cs} and N_{cd} which require four independent equations to solve, and these four independent equations do not have to exactly match Equation 13. Since we already have n_{ii}, $|S|$ and n'_{ei}, only one more equation is needed.

There are many ways to get the fourth equation. Here we present one of these methods which we found perform quite well for different datasets. The main idea is to offset the loophole effect and approximate the value of n_{ie}, or $N_d + N_o + N_{cd}$.

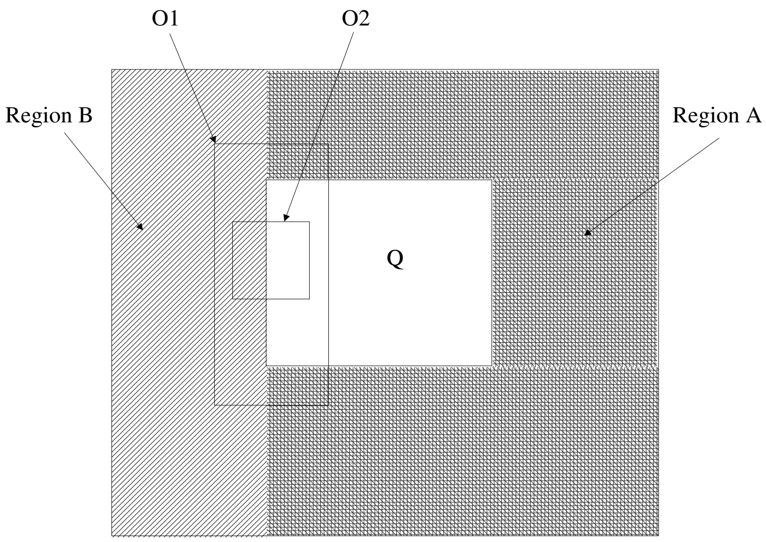


Figure 13: Estimate n_{ei}

As shown in Figure 13, given a query Q, we can divide the exterior of the query into two regions: Region A

and Region B. We first compute the number of objects that intersect Region A, and call this number $N_i(A)$. In the same way as we compute n_{ii}, the value of $N_i(A)$ can be obtained by adding up all the buckets of H that are inside Region A. Then we compute the number of objects that are contained in Region B, and call this number $N_{cs}(B)$. The value of $N_{cs}(B)$ can be computed *accurately* using the S-EulerApprox algorithm, because there is no object that can contain or "cross" Region B. Once we have $N_i(A)$ and $N_{cs}(B)$, we can use $N_i(A) + N_{cs}(B)$ to approximate n_{ie}, and the difference between these two is that objects such as $O2$ in Figure 13 are missing from $N_i(A) + N_{cs}(B)$, while objects such as $O1$ are counted twice. With H, it is not possible to determine the exact numbers of these two types of objects, but our experiments show that they tend to cancel out each other, especially when the size of the query rectangle is small.

The EulerApprox algorithm can be summarized with the following equations:

$$n_{ii} = \sum_{b_i} H(b_i) \tag{21}$$

$$n'_{ei} = \sum_{b_e} H(b_e) \tag{22}$$

$$N_o = n'_{ei} - N_d = n'_{ei} - (|S| - n_{ii}) \tag{23}$$

$$N_{cd} = N_i(A) + N_{cs}(B) - n'_{ei} \tag{24}$$

$$N_{cs} = |S| - N_{cd} - N_d - N_o \tag{25}$$

5.4 Multi-resolution Euler Approximation Algorithm

As we discussed in Section 5.3, the accuracy of the EulerApprox algorithm depends on the number of $O1$ type of objects being roughly equal to the number of $O2$ type of objects. However, observe that as the size of a query rectangle increases, the edges of the query rectangle become longer, so the possibility that an object intersects an edge of the query increases, while the possibility that an object completely contains an edge of the query decreases. In other words, the difference between the number of $O1$ type of objects and the number of $O2$ type of objects becomes larger as the query size increases, which indicates that the EulerApprox algorithm would not perform well for large queries. To address this issue, we propose the Multi-resolution Euler Approximation (M-EulerApprox) algorithm.

The idea of the M-EulerApprox algorithm is to divide the objects into multiple groups based on their areas, and construct a histogram for each group. To answer a query, we go through each of the histograms, get a partial answer by invoking either S-EulerApprox or EulerApprox algorithm based on the area of the query rectangle and the areas of the objects stored in the histogram, and combine the partial answers into the final result.

The M-EulerApprox algorithm works as follows: given a dataset and a grid, we construct m histograms H_i, where $i = 0, 1, ..., m-1$. The way to construct these histograms is the same as described in Section 5.1, except for the following:

- For each histogram H_i, we associate it with a *area* attribute, denoted as $area(H_i)$, where $area(H_i) < area(H_{i+1})$ and $area(H_0) = 1 \times 1$ (the area of the unit cell).
- H_i, where $i \neq m-1$ or 0, stores the objects whose areas are greater than or equal to $area(H_i)$, but are

smaller than $area(H_{i+1})$.

- H_{m-1} stores the objects with areas greater than or equal to $area(H_{m-1})$.
- H_0 stores the objects with areas from 0 to H_1.

So essentially, $area(H_i)$ where $i = 0, ..., m-1$ is a sequence of pre-determined values which partition the dataset into different groups, so that the objects within each group have similiar areas (from $area(H_i)$ to $area(H_{i+1})$). Finding an algorithm to determine the optimal m and the sequence $area(H_i)$ for any dataset is extremly difficult due to the fact that m and $area(H_i)$ depend on not only the areas of the objects, but also on the shapes and positions of the objects. Although an analysis based on the uniform distribution assumption is possible, we decide that it is unlikely to be useful for any practical applications. So in this paper, we introduce a pragmatic approach to determine m and $area(H_i)$ as follows:

Let $area(Q)$ be the area of a query rectangle, and assume that for a given dataset and an acceptable estimation error rate, the minimal and maximal $area(Q)$ to be supported are 1×1 and $k \times l$. We can start with 2 histograms with $area(H_0) = 1 \times 1$ and $area(H_1) = k/2 \times l/2$, and get the esimation errors for a set of test queries. If, for example, the error rate of the queries with $area(Q) < area(H_1)$ is too high, we can add another histogram H with $area(H)$ being either $area(H_1)/4$ or $area(Q)$ where at $area(Q)$ there is a peak of the estimation error rate. Repeat these steps until either the error rate for all $area(Q)$ is lower than the given limit, or adding more histograms no longer reduces the estimation error, in which case we consider the M-EulerApprox algorithm fails. Although this whole process sounds tedious, in practice it works reasonablly well because m is usually a very small number (from 2 to 5).

Given a set of histograms, we can answer queries by going through each histogram from H_{m-1} to H_0. For a query with $area(Q)$,

- if $area(Q) \leq size(H_i)$, it means that no objects in H_i that are contained in Q, or $N_{cs} = 0$. So we just need to compute the number of overlapping objects N_o. Note that both S-EulerApprox and EulerApprox use the same method to compute N_o ($N_o = \sum_{b_e} H(b_e) - (|S| - n_{ii})$), we can simply invoke S-EulerApprox.
- if $size(Q) > size(H_i)$, there are two possible cases:
    1. $size(Q) \geq size(H_{i+1})$. Since $H_{i+1} > H_i$, we can be sure that no objects in H_i can contain Q, so we invoke the S-EulerApprox algorithm and obtain N_o^i and N_{cs}^i.
    2. $size(Q) < size(H_{i+1})$ or $i = m-1$. In this case, there could be objects in H_i that contain Q, so we invoke the EulerApprox algorithm and obtain N_o^i and N_{cs}^i.

And the final results are $N_{cs} = \sum_i N_{cs}^i$, $N_o = \sum_i N_o^i$, where $i = 0, 1, ..., m$, and $N_{cd} = |S| - N_o - N_{cs}$.

5.5 Space and Time Complexity

Both S-EulerApprox and EulerApprox algorithms operate on the same histogram H, while M-EulerApprox algorithm uses multiple histogram, with each histogram having the same size as H. Given an $n_1 \times n_2$ grid, the storage space required for H is $(2n_1 - 1) * (2n_2 - 1) = O(N)$. The size of the grid depends on the resolution requirement of the browsing application.

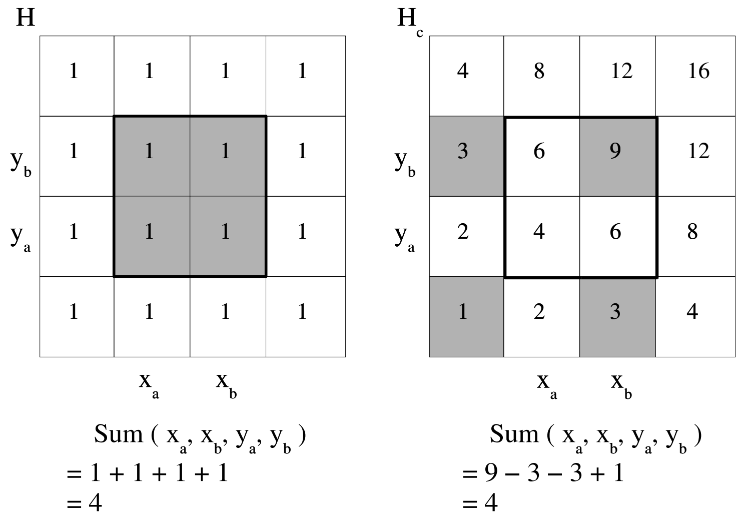


Figure 14: Cumulative Histogram H_c

For query efficiency, we use a cumulative histogram H_c instead of H, where $H_c(m,n) = \sum_{0 \le i \le m, 0 \le j \le n} H(i,j)$. An example of H with corresponding H_c is shown in Figure 14. Note that to compute the sum of the buckets in a rectangular region of H, it takes at most 4 lookups in H_c and 3 arithmetic operations, so the query response time of all three algorithms can be considered as $O(1)$.

6 Performance Evaluation

In this section we evaluate the performance of the three proposed approximation algorithms: S-EulerApprox, EulerApprox and M-EulerApprox. We use different datasets, both synthetic and real, and a variety of query sets. We are particularly interested in studying the ramifications of the different approximation assumptions made to develop these algorithms.

6.1 Experimental Setup

6.1.1 Datasets

We used four datasets in our experiments. All four datasets are spatial data in the 2-dimensional 360×180 space, all data objects are represented by their MBRs, and we use histograms at the 1×1 resolution for each of the following datasets:

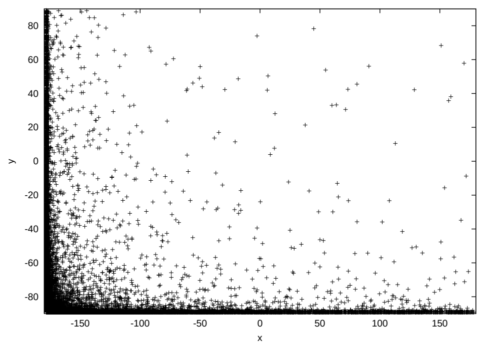
- *sp_skew* is a synthetic dataset consisting of 1 million rectangular objects, each with width 3.6 units and height 1.8 units. This dataset is designed to simulate many real world datasets which mainly consist of small objects while demonstrating significant spatial skewness. The distribution of the centers of the objects are shown in Figure 15 (a)[5].

[5]In fact, to reduce the file size (the plot output file for the complete *sp_skew* dataset is 12MB), we only plot the first 100 thousand object centers here, which should be enough to show the spatial distribution of the objects.

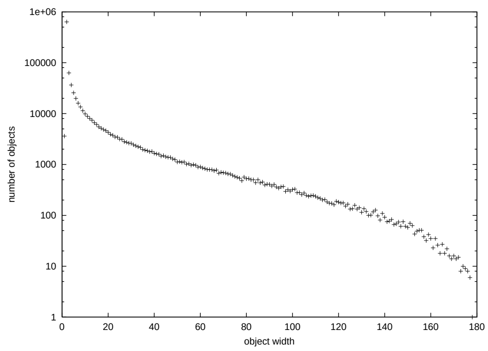
- *sz_skew* is another synthetic dataset with one million square objects. The centers of the objects are uniformly distributed in the 360×180 space. The side lengths of the objects follows a *Zipf* distribution between 1.0 and 180.0, as shown in Figure 15 (b). This dataset contains a significant number of large objects, which is rather unusual in real world scenarios, but provides a good measurement for Level 2 approximation algorithms because all three spatial relations *contains*, *contained* and *overlap* are well presented.

- *adl* is a dataset from the the Alexandria Digital Library [3]. This dataset consists of 2,335,840 objects, ranging from point data to large objects such as state, country and world maps.

- *ca_road* consists of 2,665,088 California road segments extracted from the US Census TIGER dataset [17]. All objects are normalized to the 360×180 space so we can use a consistent set of queries for all the datasets.



(a) *sp_skew* Object Center Distribution



(b) *sz_skew* Object Width Distribution

Figure 15: *sp_skew* and *sz_skew* Datasets

6.1.2 Query Sets

In our experiments, we use 11 query sets. The query sets are labeled as Q_n, where $n = 20, 18, 15, 12, 10, 9, 6, 5, 4, 3, 2$. Each query set Q_n consists of non-overlapping square queries with the same size of $n \times n$, and the queries in each query set cover the complete data space. For example, the Q_{10} query set consists of $360/10 \times 180/10 = 648$ queries of the size 10×10.

The query sets are designed to simulate browsing queries. As we discussed in Section 1, each browsing query consists of an array of "tiles", while each tiles is interpreted as a single query. So a query set Q_n is equivalent to a browsing query with the selected region being the complete data space and with tiles of the size $n \times n$.

6.1.3 Performance Metrics

We evaluate the performance of our algorithms in two aspects: *approximation accuracy* and *query processing time*.

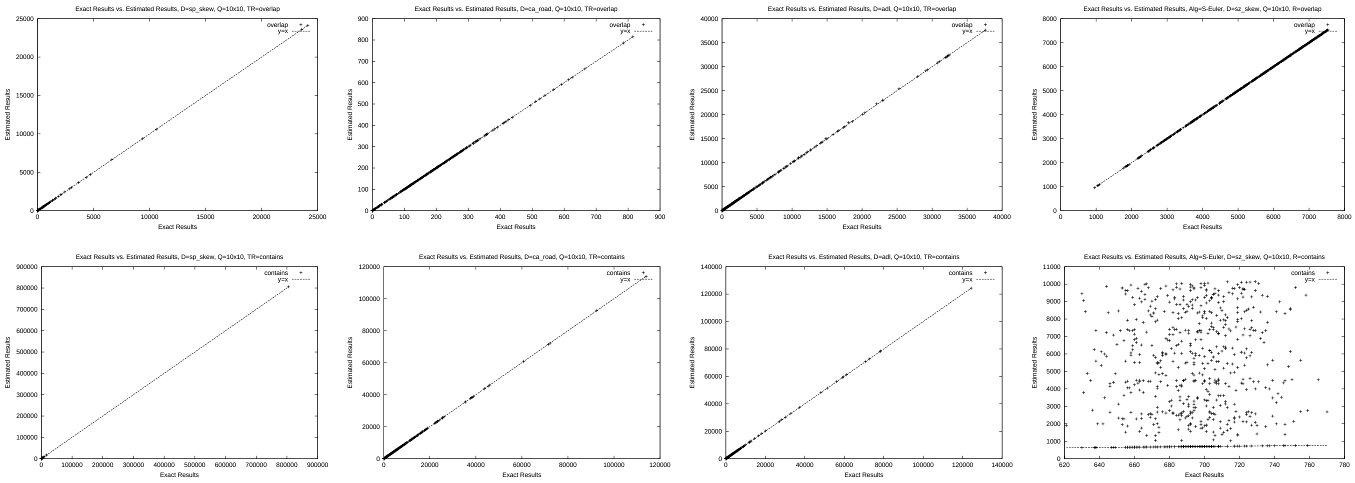
For approximation accuracy, we use *Average Relative Error* [11] as a quantitative metric. If e_i is the estimated answer and r_i is the actual answer for a given query q_i, the average relative error for a query set Q

is given as: $(\sum_{q_i \in Q} |r_i - e_i|)/(\sum_{q_i \in Q} r_i)$. For query processing time, we record the time to process each query set in wall-clock time on a PIII 800 desktop PC.

6.2 Approximation Accuracy of the S-EulerApprox Algorithm

We first evaluate the approximation accuracy of the S-EulerApprox algorithm with all four datasets. Since the S-EulerApprox algorithm assumes $N_{cd} = 0$ for all queries, we only show the results of N_o and N_{cs}.

Figure 16 shows the *overlap* results N_o and *contains* results N_{cs} for the Q_{10} query set. In this figure, the x-coordinate of a data point is the exact result of a query and the y-coordinate is the estimated result, so ideally, all data points should fall on the $y = x$ line.

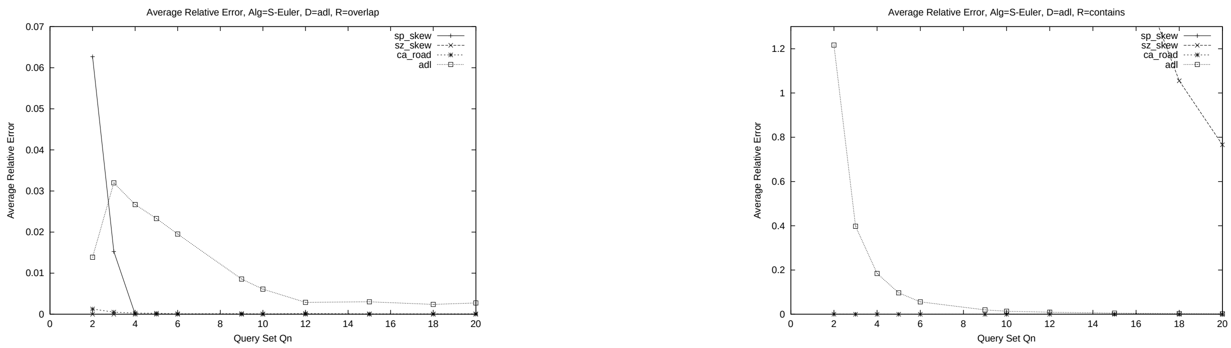


(a) *sp_skew* (b) *ca_road* (c) *adl* (d) *sz_skew*

Figure 16: N_o and N_{cs} Results of the Q_{10} Query Set

From Figure 16, we can see that S-EulerApprox works very well for the *sp_skew*, the *ca_road* and the *adl* datasets. Note that the accuracy of S-EulerApprox depends on two factors: the number of objects that contain the query, and the number of objects that "cross" the query. In this case, both of these factors are in favor of S-EulerApprox because the query size (10×10) is fairly large and the datasets consist of mostly small objects. On the other hand, for the *sz_skew* dataset, although S-EulerApprox still gives good N_o results due to small number of "cross-over" objects, it performs very badly on N_{cs}. This clearly indicates that the $N_{cd} = 0$ assumption is not applicable to the *sz_skew* dataset even for large queries.

Figure 17 (a) shows the average relative error of the *overlap* results for each query set from Q_2 to Q_{20}. Note that the accuracy of the N_o estimation is only affected by the number of crossover objects. As the query size decreases from 20×20 to 2×2, the chance of an object crossing a query generally increases, which results in a decrease of estimation accuracy. This effect is barely noticeable in the *ca_road* dataset due to its large number of small objects, but is quite evident and consistent in the *adl* dataset which contains objects with various sizes. What is very interesting and indicative are the results of the *sp_skew* and the *sz_skew* datasets. Note that the objects in the *sp_skew* dataset are of the fixed size 3.6×1.8, so a "crossover" can only occur when the query size is below 4×4. This is the reason why the error rate jump from 0 to about 1.5 percent when the query size changes from 4×4 to 3×3. In the case of the *sz_skew* dataset, the error rate of N_o is



(a) Average Relative Error of N_o (b) Average Relative Error of N_{cs}

Figure 17: Average Relative Error of the S-EulerApprox Algorithm

effectively zero, since both queries and objects are squares and it is impossible for two squares to cross each other.

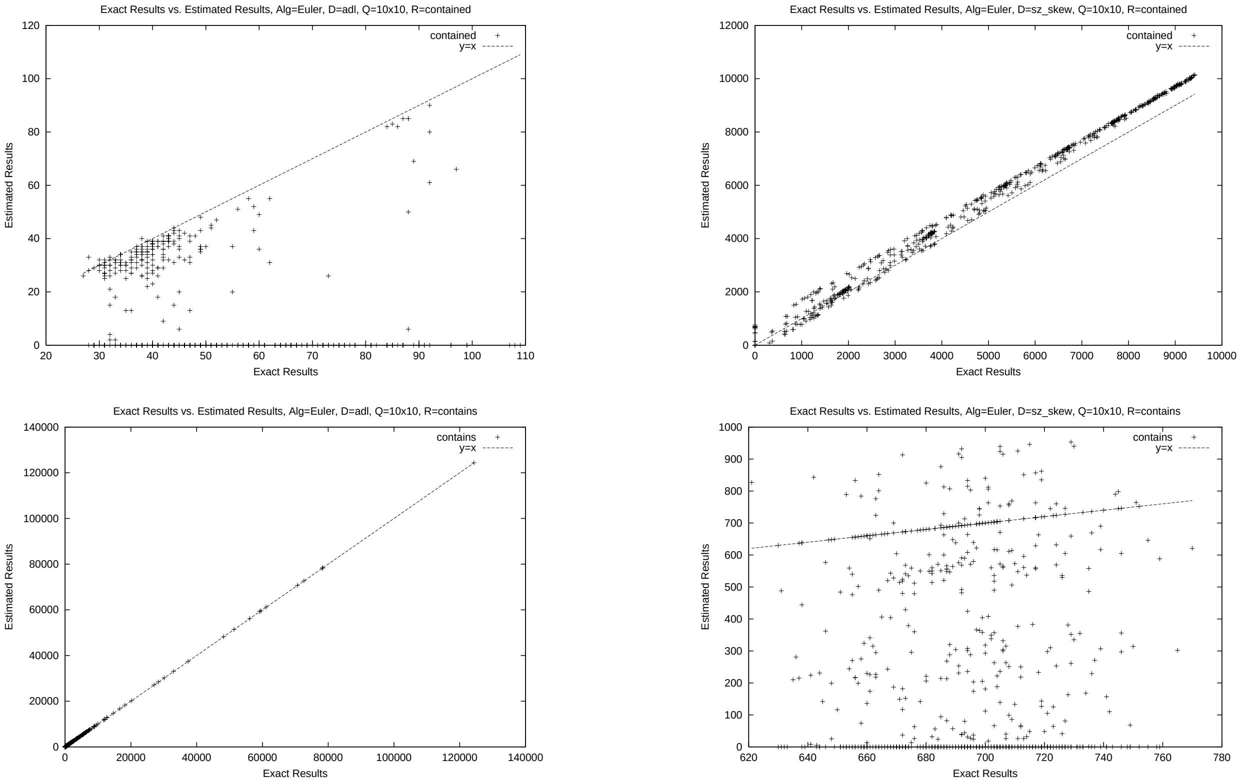
Crossover objects also affect the accuracy of the estimation of the *contains* results, but to a lesser extent than the effect of large objects. As shown in Figure 17 (b), the estimations for the *sp_skew* and the *ca_road* datasets, which consist mostly of small objects, are very accurate for all query sizes. However, the error rate of the *sz_skew* dataset go out of chart even for large query sizes, while the error rate of the *adl* also increases rapidly as the query size decreases. Note that the S-EulerApprox algorithm assumes that the number of objects that contains the query, or N_{cd}, is sufficiently small compared to N_o and N_{cs}. Apparently, this assumption is not valid for the *adl* or the *sz_skew* dataset which contains a significant number of large objects. As the query size decreases, the number of objects that are contained in a query becomes smaller, while the number of objects that contain the query becomes larger. The combined effect is that the the error rate increases almost exponentially.

6.3 Approximation Accuracy of the EulerApprox Algorithm

In this section we evaluate the EulerApprox algorithm, which takes large objects into consideration. Since the S-EulerApprox algorithm already provides good approximation results for the *sp_skew* and the *ca_road* datasets, in this experiments we only consider the *adl* and the *sz_skew* datasets. Also since all three approximation algorithms S-EulerApprox, EulerApprox and M-EulerApprox use exactly the same method to estimate N_o, the N_o results are omitted in this section and in Section 6.4.

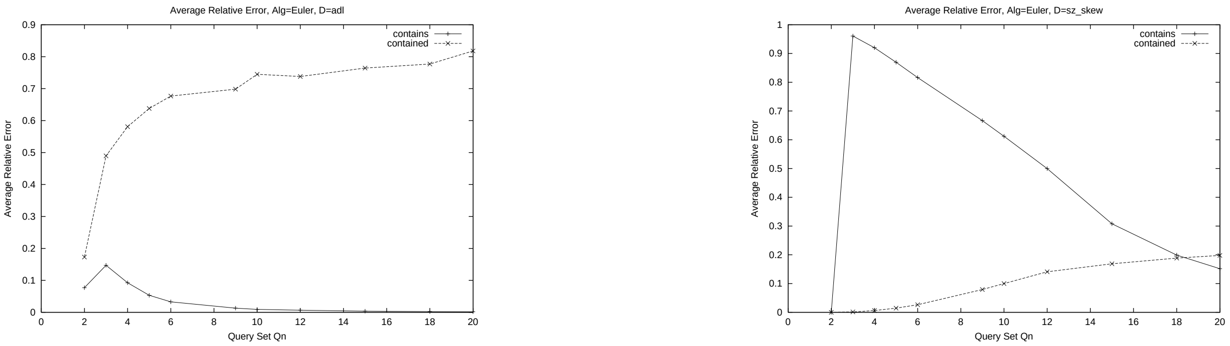
Figure 18 shows the estimated N_{cd} and N_{cs} results vs. the exact results of the Q_{10} query set. As we can see, for the *adl* dataset, the EulerApprox algorithm is not doing a good job on estimating N_{cd}, but the N_{cs} results are still very good. The situation is reversed for the *sz_skew* dataset, where the N_{cd} estimation is reasonablly accurate but the N_{cs} results are quite bad. Figure 18 seems to contradict the intuition that given an accurate N_o estimation, the more accurate the N_{cd} estimation is, the more accurate the N_{cs} estimation will be. However, on a closer look, we will see that for the *adl* dataset, the N_{cs} values are orders of magnitude larger than the N_{cd} results, which means that the N_{cs} results are very resilient to N_{cd} estimation errors; while in the case of the *sz_skew* dataset, the values of N_{cd} is about an order of magnitude larger than the values of N_{cs}, so about 10% error in N_{cd} completely dominates N_{cs}.

Figure 19 shows the average relative error of the EulerApprox algorithm. Note that EulerApprox assumes that the $O1$ type of objects (see Figure 13) are of the same number as the $O2$ objects, which is clearly a



(a) *adl* (b) *sz_skew*

Figure 18: N_{cd} and N_{cs} Results of the Q_{10} Query Set



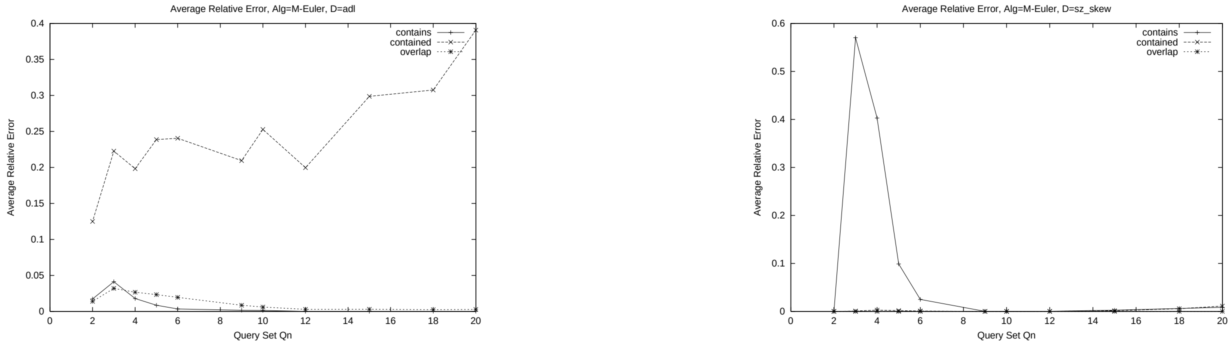
(a) *adl* (b) *sz_skew*

Figure 19: Average Relative Error of the EulerApprox Algorithm

very simplistic assumption. However, by making this very simplistic assumption, the accuracy of N_{cs}, which is often considered as a more important metric in practice, improves noticeably. Comparing Figure 19 with Figure 17 (b), we can see that for the *adl* dataset, the worst case N_{cs} error rate drops from 120% to a somewhat tolerable 15%; and for the *sz_skew* dataset, although the N_{cs} error rate is still quite high, it is still a great improvement comparing to the error rate in Figure 17 (b). Overall, the EulerApprox algorithm is a big improvement over the S-EulerApprox algorithm, but the end results are still not satisfactory.

6.4 Approximation Accuracy of the M-EulerApprox Algorithm

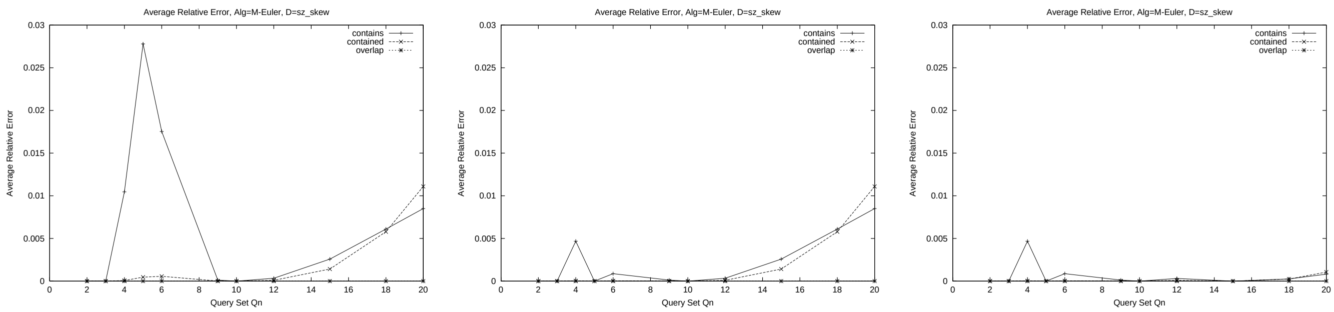
In this section we evaluate the M-EulerApprox algorithm on the *adl* and the *sz_skew* datasets, and see if trading storage space can further improve the accuracy of the N_{cs} and N_{cd} estimations.



(a) *adl* (b) *sz_skew*

Figure 20: Average Relative Error of the M-EulerApprox Algorithm with 2 Histograms

Figure 20 shows the average relative error of the M-EulerApprox algorithm with 2 histograms, where $size(H_0) = 1 \times 1$ and $size(H_1) = 10 \times 10$. Comparing Figure 20 to Figure 19, we can see that by simply adding one additional histogram, the esitmation accuracy improves dramatically. For the *adl* dataset, the worst case N_{cs} error rate is now less than 5%. For the *sz_skew* dataset, both N_{cs} and N_{cd} estimations are highly accurate for large query sizes, but the N_{cs} accuracy is still unsatisfactory for small query sizes.



(a) 3 histograms (b) 4 Histograms (c) 5 Histograms

Figure 21: Average Relative Error of the M-EulerApprox Algorithm for the *sz_skew* Dataset

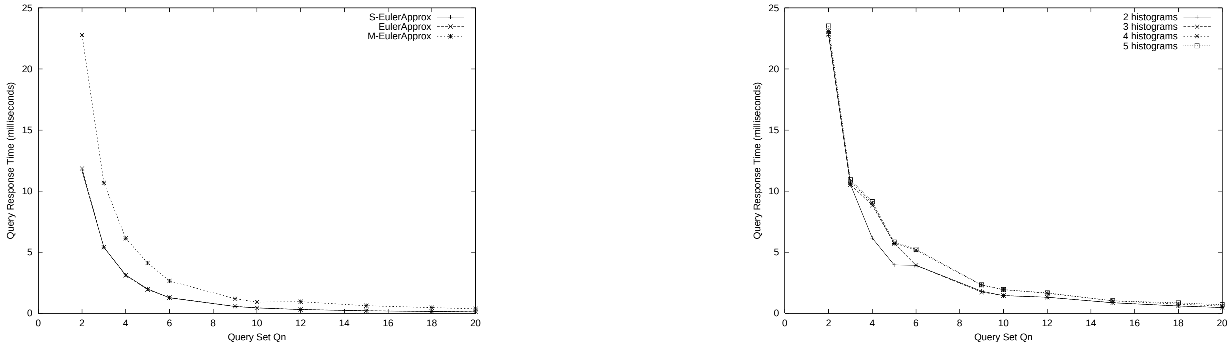
To further improve the estimation accuracy for the *sz_skew* dataset, we increase the number of histograms used in the M-EulerApprox algorithm, and the results are shown in Figure 21. The $area(H_i)$ values in the three experiments shown in Figure 21 are:

- 3-histogram case: $area(H_i) = 1 \times 1$, 3×3 and 10×10
- 4-histogram case: $area(H_i) = 1 \times 1$, 3×3 and 5×5, 10×10
- 5-histogram case: $area(H_i) = 1 \times 1$, 3×3 and 5×5, 10×10, 15×15

As we can see, with 3 histograms, the worst case error rate already drops from about 58% to below 3% (note the difference between the y-axis scales in Figure 20 (b) and Figure 21). But maybe more importantly, as the number of histograms increase, the estimation accuracy consistently improves.

6.5 Query Response Time

Theoretically, both S-EulerApprox and EulerApprox algorithms take constant time to answer a single query. We measured the query response time for each query set and give some quantitative results in Figure 22. Note that the size of a query set Q_n can be calculated by $360/n \times 180/n$. For example, the Q_2 query set consists of $360/2 \times 180/2 = 16200$ queries.



(a) Query Processing Time of Different Algorithms (b) Query Processing Time of M-EulerApprox Algorithm

Figure 22: Query Processing Time

As we can see, all three algorithms take less than 25 milliseconds to processes the largest query set, so all of them are efficient enough for browsing applications[6]. One thing worth noting is that the time difference between S-EulerApprox and EulerApprox is almost negligible. This is because the query processing time of both algorithms are dominated by computing the indexes of the histogram H from the query rectangle. This computation involves floating point arithmetic and branch statements, which are much more expensive than lookups and integer operations on modern processors. Another somewhat surprising result is that the query processing time is roughly the same for the M-EulerApprox algorithm regardless of the number of the histograms used. This may also be due to the fact that the most expensive operation, namely, the index computation, are only done once for all histograms.

7 Conclusion

Spatial database browsing is an important problem that has not been systematically studied before. In this paper, we concentrated on the efficient computation of Level 2 spatial relations which need to be supported in

[6]The goal we started out with was to process a browsing query with 5000 tiles under 0.1 seconds.

browsing applications. By extending the spatial relations that can be efficiently handled from Level 1 to the Level 2, we are able to open up some new application possibilities. We proposed the interior-exterior model, which presents a new perspective on the relationship between queries and objects. This allows us to explore types of queries like *contains* which were not handled by prior approaches. Under this model, we proved that exact evalution of Level 2 queries requires substantial storage overhead. Based on this model and Euler's Formula, we developed three storage-efficient approximation algorithms with constant time complexity. The performance evaluation shows that the S-EulerApprox algorithm achieve high approximation accuracy for datasets that are dominated by small objects. For datasets in which the number of large objects is significant, the M-EulerApprox performs very well with slightly increased time and space complexity.

References

[1] Volker Gaede and Oliver Günther. Multidimensional access methods. *Computing Surveys*, 30(2):170–231, 1998.

[2] Douglas M. Flewelling and Max J. Egenhofer. Using digital spatial archives effectively. *International Journal of Geographical Information Science, 13 (1): 1-8, 1999*, 1999.

[3] UC Santa Barbara. Alexandria digital library project. *http://www.alexandria.ucsb.edu*, 1999.

[4] Stephan Greene, Egemen Tanin, Catherine Plaisant, Ben Shneiderman, Lola Olsen, Gene Major, and Steve Johns. The end of zero-hit queries: Query previews for nasa's global change master directory. *International Journal of Digital Libraries*, pages 79–90, February 1999.

[5] R. Beigel and Egemen Tanin. The geometry of browsing. In *Proceedings of the Latin American Symposium on Theoretical Informatics, 1998, Brazil*, pages 331–340, 1998.

[6] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. Data cubes: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. *Data Mining and Knowledge Discovery*, 1(1), 1997.

[7] Ching-Tien Ho, Rakesh Agrawal, Nimrod Megiddo, and Ramakrishnan Srikant. Range queries in olap data cubes. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, pages 73–88, May 1997.

[8] Chee Yong Chan and Yannis E. Ioannidis. Hierarchical prefix cubes for range-sum queries. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases*, pages 675–686, September 1999.

[9] Steve Geffner, Mirek Riedewald, Divyakant Agrawal, and Amr El Abbadi. Data cubes in dynamic environments. *Data Engineering Bulletin*, 22(4):31–40, 1999.

[10] Mirek Riedewald, Divyakant Agrawal, and Amr El Abbadi. pcube: Update-efficient online aggregation with progressive feedback and error bounds. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 95–108, 2000.

[11] Swarup Acharya, Viswanath Poosala, and Sridhar Ramaswamy. Selectivity estimation in spatial databases. In *SIGMOD'99, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 13–24, June 1999.

[12] Ashraf Aboulnaga and Jeffrey F. Naughton. Accurate estimation of the cost of spatial selections. In *ICDE'00, Proceedings of the 16th International Conference on Data Engineering*, pages 123–134, March 2000.

[13] Ji Jin, Ning An, and Anand Sivasubramaniam. Analyzing range queries on spatial data. In *ICDE'00, Proceedings of the 16th International Conference on Data Engineering*, pages 525–534, March 2000.

[14] Max J. Egenhofer and John R. Herring. Categorizing binary topological relations between regions, lines, and points in geographic databases. In Max J. Egenhofer, David M. Mark, and John R. Herring, editors, *The 9-Intersection: Formalism and Its Use for Natural-Language Spatial Predicates. National Center for Geographic Information and Analysis, Report 94-1*, pages 13–17, 1994.

[15] Frank Harary. *Graph Theory.* Addison-Wesley Publishing Company, 1969.

[16] William S. Massey. *Algebraic Topology: An Introduction.* Brace & World, 1967.

[17] TIGER. 1997 TIGER/Line Files (machine-readable data files). Technical report, U.S. Bureau Of the Census, Washington, DC, 1997.