Wavelet-Based Cost Estimation for Spatial Queries

Min Wang[*] *Jeffrey Scott Vitter*[†] *Lipyeow Lim*[‡] *Sriram Padmanabhan*[§]

Abstract

Query cost estimation is an important and well-studied problem in relational database systems. In this paper we study the cost estimation problem in the context of spatial database systems.

We introduce a new method that provides accurate cost estimation for spatial selections, or window queries, by building wavelet-based histograms for spatial data. Our method is based upon two novel techniques: (a) A representation transformation in which geometric objects are represented by points in higher-dimensional space and window queries correspond to semi-infinite range-sum queries, and (b) Multiresolution wavelet decomposition that provides a time-efficient and space-efficient approximation of the underlying distribution of the multidimensional point data, especially for semi-infinite range-sum queries. We also show for the first time how a wavelet decomposition of a dense multidimensional array derived from a sparse array through a partial-sum computation can still be computed efficiently using sparse techniques by doing the processing implicitly on the original data. Our method eliminates the drawbacks of the partition-based histogram methods in previous work, and even with very small space allocation it gives excellent cost estimation over a broad range of spatial data distributions and queries.

[*] Contact author. IBM T. J. Watson Research Center, 30 Saw Mill River Road, Hawthorne, NY 10532. Phone: (914) 784–6268, FAX: (914) 784–6079, Email: min@us.ibm.com

[†] Center for Geometric Computing, Department of Computer Science, Duke University, Durham, NC 27708–0129. Email: jsv@cs.duke.edu

[‡] Center for Geometric Computing, Department of Computer Science, Duke University, Durham, NC 27708–0129. Email: lipyeow@cs.duke.edu

[§] IBM T. J. Watson Research Center, 30 Saw Mill River Road, Hawthorne, NY 10532. Email: srp@us.ibm.com

1 Introduction

Spatial data appear in numerous applications, such as GIS, multimedia, and even traditional databases. DBMSs (DataBase Management Systems) must offer spatial query processing capabilities to meet the needs of such applications. Query optimization is an integral part of DBMSs. One important task in query optimization is query cost estimation. In this paper, we study the cost estimation problem for an important class of spatial queries, *spatial selections* (or *window queries*).

In a window query, a region called *query window* is specified, and the query retrieves all the objects in the data set that partially or completely overlap the region. Spatial objects can be very complex, and their accurate representation may require a large amount of memory. Since manipulating such large objects during query optimization time can be very expensive, it is customary to approximate spatial objects and manipulate the approximations instead. The most common techniques is to bound each spatial object by the smallest axis-parallel rectangle that completely contains it, called *minimum bounding rectangle* (MBR).

Database systems process window queries and other spatial operations using a two-step *filter and refine* strategy [Ore86]. The filtering step identifies the set of *candidate objects* whose MBRs partially or completely overlap the MBR of the query window. The refinement step tests the exact geometry of the the candidate objects identified by the filtering step to determine the set of objects that actually overlap the query window. Thus, the cost of window query depends upon the selectivity of the filtering step (the *MBR selectivity*) and the complexity of the candidate objects [AN00].

The start-of-the-art techniques on spatial cost estimations [AN00, APR99] deal with the spatial objects in their original space directly and form partition-based histograms for spatial data using different partition strategies. The goal of any partition strategy is to form histogram buckets in a way that each bucket contains objects that are similar in each of the feature domains (e.g., location, shape). However, because of the complicated nature of spatial objects, such a goal is often not achievable in practice.

In this paper, we take a different approach. We first use a transformation of representation to transform the spatial objects into points in higher-dimensional space. All the features of the objects in the original space can be reflected as the frequency distribution of the points in the transformed space. Window queries correspond to semi-infinite range-sum queries in the transformed space. We then use a novel multiresolution compression technique, namely, a type of wavelet decomposition done on the partial sums (prefix/suffix sums) of the transformed data, to form a histogram for the point data. The partial sums correspond exactly to the window queries, and thus the approximation should be especially good. Experiments confirm that wavelet decomposition on the partial sums of the transformed data yields noticeably better results than does wavelet decomposition directly on the transformed data. One problem is that the partial sum array is very dense and infeasible to be processed by conventional means.

Our main contributions are as follows:

1. Our approximation method is robust over a broad range of spatial data distributions and queries. Even with very small space allocation it gives excellent cost estimation.

2. Our method provides a uniform framework of doing spatial cost estimations in spaces of different dimensionalities. While we concentrate on two-dimensional space in this paper, our method can be applied to one-dimensional and three dimensional space as well.
3. Our method can be used to answer spatial queries approximately and support progressive refinement of the approximate answers efficiently. While partition-based histogram methods can also be used for approximate query processing, they can only be applied in a static way and thus do not have the desirable feature of supporting progressive refinement.
4. Although the transformed data set is typically sparse when represented in a discrete multidimensional array, in that most values in the array are zeroes, the array consisting of partial sums of the transformed data, which is most effective for our wavelet decomposition in terms of approximation, is very dense and has few nonzero values. Thus, the partial sum wavelet decomposition cannot be processed directly in a space-efficient manner. This limitation has been a major stumbling block in the applicability of partial sums for query approximation in previous work on wavelets [VW99]. It is particularly important for us to resolve this problem since partial sums are especially well suited to window queries. We solve this open problem by computing the wavelet decomposition of the dense partial sum array using an indirect approach in which sparse techniques can be applied directly to the sparse transformed array to compute the partial sums and the wavelet decomposition simultaneously.

The rest of the paper is organized as follows: In the next section we summarize related work. We formally define the problem in Section 3. In Sections 4 and 5, we describe our method. We present our experimental results in Section 6 and draw conclusions in Section 7.

2 Related Work

Selectivity estimation is a well-studied problem for traditional data types such as integers. Histograms are the most widely used form for doing selectivity estimation in relational database systems. Many different histograms have been proposed in the literature [PSC84, PIHS96, MD88, PI97] and some have been deployed in commercial RDBMSs. However, almost all previous histograms have one thing in common, that is, they use buckets to partition the data, although in different ways.

In [MVW98] Matias et al. introduce a new type of histograms, called *wavelet-based histograms*, based upon a multidimensional wavelet decomposition. A wavelet decomposition is performed on the underlying data distribution, and the most significant wavelet coefficients are chosen to compose the histogram. In other words, the data points are "compressed" into a set of numbers (or wavelet coefficients) via a sophisticated multiresolution transformation. Those coefficients constitute the final histogram. This approach offers more accurate selectivity estimation than traditional partition-based histogram methods and can be extended very naturally to efficiently compress the joint distribution of multiple attributes. Experiments confirm that wavelet-based histograms offer more accurate selectivity estimation for typical queries than traditional bucket-partition based histogram methods.

Cost estimation in spatial databases is a relatively new topic, and some techniques for window queries have been proposed in the literature [FK94, APR99, KF93, PSTW93, BF95, AN00, TS96]. With the exception of [AN00], all the previous estimation techniques assume that the query windows and data objects are rectangles. They estimate the selectivity and the cost of the filtering step, and they ignore the refinement step.

Aboulnaga and Naughton [AN00] introduce an interesting new type of histogram for general polygon data and query windows. The new histogram, called the SQ-histogram, partitions the data objects into possible overlapping rectangular buckets. The partitioning is based upon the MBRs of the objects, with each object being assigned to one histogram bucket. Each bucket stores the number of objects it represents, the objects' average width and height, and their average number of vertices, as well as the boundaries of the rectangular region containing these objects. The objects within a bucket are assumed to be uniformly distributed. The information about the average number of vertices in each bucket is important for estimating the cost of the refinement step, and thus this work is quite different from previous works that estimate only the selectivity of the filtering step.

While the SQ-histogram is a novel technique for capturing spatial data distributions, it still falls into the category of partition-based histograms, and it cannot overcome the limitations that partition-based histograms have, especially when dealing with spatial objects. Since assuming uniformity within each bucket is the major reason that introduces estimation errors for partition-based histograms, the goal of histogram construction is to form buckets in a way that each bucket represents a "homogeneous" set of objects. Because of the strict storage size constraints for histograms used in query optimization, the number of buckets is usually very small. Using a small number of buckets to approximate the underlying distribution so that each bucket contains data points with "similar" frequencies is a hard job even for traditional partition-based histograms used for simple point data. For spatial data, we require uniformity within each bucket in several domains: location, width of the MBRs, height of the MBRs, complexity of the objects. It is extremely difficult to find a partition mechanism to ensure that the uniformity holds for all these domains unless we use a large number of buckets! As a result, partition-based histograms often do not provide accurate cost estimation with low storage overhead.

3 Problem Formulation

In this section, we formally define the cost estimation problem addressed in this paper. Consider a relation R containing an attribute whose domain is a set of N general polygons. The set of polygons can be approximately represented as

$$A = \{\big((x_l^i, y_b^i), (x_r^i, y_t^i), v^i\big) \mid 1 \leq i \leq N\},$$

where (x_l^i, y_b^i) and (x_r^i, y_t^i) specify the lower-left and upper-right corner of the MBR of the ith polygon, respectively, and v^i is the number of vertices of the ith polygon. A window query Q is a polygon whose MBR can be represented as

$$q = \big((Qx_l, Qy_b), (Qx_r, Qy_t)\big).$$

We define the set of objects whose MBRs partially or completely overlap the query MBR q as the *candidate set*:

$$C = \{c_i \mid c_i \in A \text{ and } c_i \text{ overlaps } q\}.$$

The *MBR selectivity* of query Q is defined as

$$s_{MBR} = \frac{1}{N}|C|.$$

The *complexity* of the candidate set is defined as

$$c_{cand} = \frac{1}{|C|}\sum_{c_i \in C} v^i.$$

That is, the complexity of the candidate set is the average number of vertices of the candidate objects.

Aboulnaga and Naughton [AN00] showed that approximating the cost of a window query requires estimating the MBR selectivity s_{MBR} of the query and the complexity c_{cand} of the candidate set. However, computing these two quantities exactly requires us to either scan the data set or use an existing index to find each member of the candidate set. Both of these two options are too expensive to be useful in query optimization since an essential requirement for a query optimizer is to do cost estimation for any query in a time and space efficient manner. In a DBMS, cost estimation is usually done through two phases: an *offline phase* and an *online phase*. During the offline phase, the input data are approximated and some statistical/summary information is collected and stored in the *database catalog*. The space allocated in the catalog for one attribute of a relation is usually very small (often in the order of several hundred bytes in traditional RDBMSs). During online phase, the cost of a given query is estimated based upon the relevant catalog information.

The goal of our work is to construct accurate histograms for spatial data and use it to do cost estimation (i.e., to estimate s_{MBR} and c_{cand}) for any window query efficiently.

4 Constructing Wavelet-Based Histograms for Spatial Data

At a high level, our cost estimation model works in four steps during the offline phase:

1. *Transformation:* We transform the input spatial objects into points in higher-dimensional space.
2. *Quantization and Preprocessing:* We form a uniform grid for the point data through quantization and compute partial sums of the grid data.
3. *Wavelet Decomposition:* We compute the wavelet decomposition of the grid data, obtaining a set of wavelet coefficients.

4. *Thresholding:* We keep only the m most significant wavelet coefficients, for some m that corresponds to the desired storage usage. The choice of which m coefficients we keep depends upon the particular thresholding method we use. The m coefficients compose our wavelet-based histogram.

In Sections 4.1–4.4, we elaborate on the details of these four steps.

For large GIS data, the number of objects may be very large and typically cannot be stored in internal memory, even after being transformed into points. It is convenient to treat the points in a discrete way by dividing the space into a uniform grid of four-dimensional rectangular *grid regions*. Even though the number of points (objects) may be large, their number is typically very small with respect to the number of grid regions. *That is, the input data is typically very sparse.* Thus it may not be possible to process the full grid at all, even using efficient external memory techniques [Vit99], and thus sparse techniques are needed. As we shall see, a potentially major stumbling block is that it is not possible to use known sparse techniques, such as those recently developed by Vitter and Wang [VW99], because of the partial sum preprocessing, which makes the grid array highly dense. In Section 4.3 we show nevertheless how to do the sparse wavelet decomposition in an I/O-efficient manner.

4.1 Transformation of Representation

Transforming the representation of the input data is a well-known technique for representing geometric objects [NH85, SK88, WKW94]. The idea is to transform objects in the original space (*o-space*) into points in the transformed space (*t-space*) using parameters that represent the shape and location of each object. The transformed points can then be accessed by a multidimensional point access method during spatial query processing.

The transformation technique can certainly be applied to cost estimation in spatial databases. After transforming geometry objects into points in higher-dimensional space, we can use various multi-dimensional histogram methods that have been developed for point data in traditional databases [MD88, PI97, MVW98, VWI98, Wan99, VW99]. The combination of proper transformation and an accurate multidimensional histogram method can result in very accurate cost estimation for spatial operations. However, to the best of our knowledge, no previous work on spatial cost estimation has ever considered applying a representation transformation.

In our method, we use a *corner transformation* to represent the MBR of a spatial object. For simplicity, we first explain how to apply corner transformation in a one-dimensional o-space. Corner transformation maps an object in the one-dimensional o-space into a point in the two-dimensional t-space. The left end of an object is mapped to the horizontal (x_l) axis and the right end to the vertical (x_r) axis in the t-space. All transformed objects are placed in the upper part of the diagonal in the t-space since the x_r value (the coordinate of the right end) of an object is always larger than the x_l value (the coordinate of the left end). Figure 1 shows the procedure of transforming a set of intervals in the o-space into a set of points in the t-space.

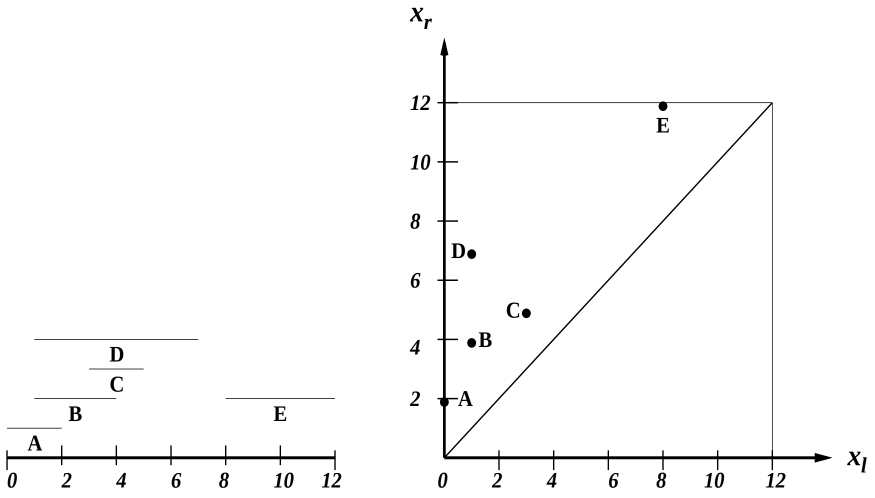


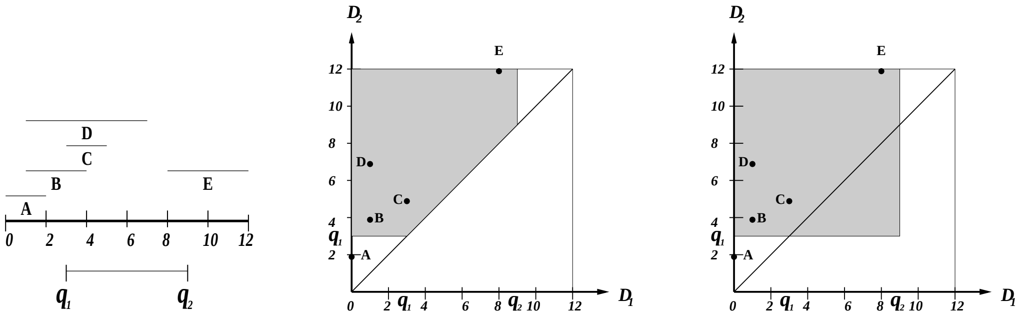
Figure 1: A set of intervals and its corner transformation

For cost estimation of window queries, we use corner transformation to map the MBR of a polygon in the two-dimensional o-space to a point in the four-dimensional t-space. We name the four axes of the t-space as x_l, x_r, y_b, and y_t. A rectangle with lower-left corner (x_l^i, y_b^i) and upper-right corner (x_r^i, y_t^i) in the o-space is transformed into a point $(x_l^i, x_r^i, y_b^i, y_t^i)$ in the t-space.

The nice property of the transformation is that the coordinates of a point in the t-space capture all the features of the corresponding MBR in the o-space in a very compact way. We can calculate the width and the height of a rectangle easily using its point representation. Also the relationship among objects becomes very obvious in the t-space. For example, two points in the t-space are close to each other if and only if the two corresponding MBRs in the o-space are both very close in location and similar in shape (i.e., similar in both width and height).

4.2 Quantization, Preprocessing, and the Partial Sum Array

To discretize the problem, we quantize the t-space evenly into a uniform grid of four-dimensional rectangular *grid regions*. Let $D = \{D_1, D_2, D_3, D_4\}$ denote the set of four dimensions in the quantized t-space, where dimensions D_1, D_2, D_3 and D_4 correspond to x_l, x_r, y_b and y_t, respectively. We can approximately represent the objects in the o-space by a four-dimensional grid array of size $|D_1| \times |D_2| \times |D_3| \times |D_4|$, where $|D_i|$ is the size of dimension D_i. Note that we always have $|D_1| = |D_2|$ and $|D_3| = |D_4|$. Without loss of generality, we assume that each dimension D_i has an index domain $\{0, 1, \ldots, |D_i| - 1\}$. For convenience, we call each array element a *grid cell*. A cell indexed by (i_1, i_2, i_3, i_4) contains two values, $p(i_1, i_2, i_3, i_4)$ and $v(i_1, i_2, i_3, i_4)$, where $p(i_1, i_2, i_3, i_4)$ is the number of points in the corresponding four-dimensional rectangular region, and $v(i_1, i_2, i_3, i_4)$ is the total number of vertices for the objects counted in $p(i_1, i_2, i_3, i_4)$. Let $G = \prod_{1 \le i \le 4} |D_i|$ denote the total size (i.e., number of grid cells) of the grid array. For convenience, we call the domain of possible $p(i_1, i_2, i_3, i_4)$ values the *p-domain*, and we call the domain of possible $v(i_1, i_2, i_3, i_4)$ values the



(a) query and data in o-space

(b) query and data in t-space

(c) query and data in extended t-space

Figure 2: Query region and data.

v-domain.

Now we examine how to use the two array representations to calculate the MBR selectivity and complexity of any given window query. For simplicity, let us first consider the MBR selectivity in a one-dimensional o-space. We can (approximately) represent a one-dimensional query interval by a cell (i_1, i_2) in the two-dimensional grid space. A given query MBR q can be (approximately) represented as a cell (q_1, q_2) in the grid space. The MBR selectivity of q is the fraction of the intervals that partially or completely overlap with q. We can easily see that an interval in the o-space overlaps with q if and only if its transformed point in the t-space is in the shaded area in Figure 2b. Since there is no point below the diagonal in the t-space, we can also say that an interval in the o-space overlaps with q if and only if its transformed point in the t-space is in the extended shaded area in Figure 2c. Thus, we can use the following formula to calculate the MBR selectivity of query q:

$$s_{MBR} = \frac{1}{N} \sum_{0 \le i_1 \le q_2} \sum_{q_1 \le i_2 < |D_2|} p(i_1, i_2). \tag{1}$$

In a two-dimensional o-space, for any given query window q whose MBR can be (approximately) represented as (q_1, q_2, q_3, q_4) in the grid space, the MBR selectivity and complexity of q can be calculated using the following formulas:

$$s_{MBR} = \frac{1}{N} \sum_{0 \le i_1 \le q_2} \sum_{q_1 \le i_2 < |D_2|} \sum_{0 \le i_3 \le q_4} \sum_{q_3 \le i_4 < |D_4|} p(i_1, i_2, i_3, i_4), \tag{2}$$

$$c_{cand} = \frac{\sum_{0 \le i_1 \le q_2} \sum_{q_1 \le i_2 < |D_2|} \sum_{0 \le i_3 \le q_4} \sum_{q_3 \le i_4 < |D_4|} v(i_1, i_2, i_3, i_4)}{\sum_{0 \le i_1 \le q_2} \sum_{q_1 \le i_2 < |D_2|} \sum_{0 \le i_3 \le q_4} \sum_{q_3 \le i_4 < |D_4|} p(i_1, i_2, i_3, i_4)}. \tag{3}$$

For queries whose MBR do not correspond exactly to a grid point, we use an interpolated version of (2) and (3).

Since formulas (2) and (3) are in the form of partial sums, we preprocess arrays p and v to obtain two new arrays P and V so that

$$P(i_1, i_2, i_3, i_4) = \sum_{0 \le j_1 \le i_1} \sum_{i_2 \le j_2 < |D_2|} \sum_{0 \le j_3 \le i_3} \sum_{i_4 \le j_4 < |D_4|} p(j_1, j_2, j_3, j_4), \tag{4}$$

and

$$V(i_1, i_2, i_3, i_4) = \sum_{0 \le j_1 \le i_1} \sum_{i_2 \le j_2 < |D_2|} \sum_{0 \le j_3 \le i_3} \sum_{i_4 \le j_4 < |D_4|} v(i_1, i_2, i_3, i_4). \tag{5}$$

That is, we compute the prefix sums along dimensions D_1 and D_3 and the suffix sums along dimensions D_2 and D_4 for both array p and array v to obtain two new arrays P and V, respectively. Using the new arrays P and V, the formulas (2) and (3) can be written as:

$$s_{MBR} = \frac{1}{N} P(q_2, q_1, q_4, q_3), \tag{6}$$

$$c_{cand} = \frac{V(q_2, q_1, q_4, q_3)}{P(q_2, q_1, q_4, q_3)}. \tag{7}$$

We have experimented with the approach of using partial sums as described above in formulas (4) and (5) versus the approach of not using partial sums. For our purposes, the smoothness due to partial sums in the arrays P and V intuitively enables a more effective wavelet approximation and as a result gives better results than the wavelet approximation on p and v. In the next subsections we discuss the wavelet decomposition on P and V and the subsequent thresholding, and we show how the processing can be done in an I/O-efficient manner despite the fact that P and V are dense arrays.

4.3 I/O-Efficient Wavelet Decomposition of Dense Partial Sum Array

The array representations of P and V are often too big to fit in internal memory or even on disk when the dimensionality of the array is 4 or greater, and they are certainly much too big to be stored in a database catalog. The goal of this step is to use wavelets as an efficient compression tool to construct a good approximate representation of P and V.

We refer readers who are not familiar with wavelet decomposition procedure to Section 5.1 of [VW99]. For illustration purposes, we use the Haar wavelets described there throughout this paper.

The main challenge for this step is how to perform the wavelet decomposition on the partial-sum array representations of P and V in an I/O-efficient manner. The original data (i.e., before partial sums are formed) are often very sparse with respect to the large grid array. When the dimension is 4 or greater, the grid array may be too large to store explicitly even on disk without using some sparse represnetation (like the one suggested in [VW99]). When partial sums are formed, the grid array becomes very dense. A single nonzero value in the original data could cause each entry in the partial sum array to be nonzero. Wavelet decomposition is not feasible in such a scenario by conventional techniques. This limitation has been a major limiting factor in the applicability of partial sums for query approximation in previous work on wavelets [VW99]. It is particularly important for us to resolve this problem since partial sums are especially well suited to window queries.

We solve this open problem by computing the wavelet decomposition of the dense partial sum array by using an indirect approach in which sparse techniques can be applied. The intuition is that the final wavelet decomposition itself is sparse. And thus it may be possible to process the original sparse data directly and to realize the final sparse wavelet decomposition of the partial sum array. To accomplish this goal, we cycle through the dimensions, and for each dimension we do the partial sum calculation at the same time that the wavelet decomposition is done for that dimension. At any given point in the processing, some number of dimensions have been decomposed, and the "slices" of the multidimensional array corresponding to those dimensions are therefore sparse. The partial sums along the remaining dimensions have not yet been formed, so they remain sparse as well. Details and full analysis will be included in the full paper.

Theorem 1 *The number of I/Os needed to do a d-dimensional wavelet decomposition of size $G = |D_1| \times |D_2| \times \cdots \times |D_d|$ with internal memory of size M and block size B is*

$$O\left(\frac{N_z}{B}\min\left\{d,\, \log_{M/B}\frac{G}{B}\right\}\right),$$

where N_z is the number of nonzero coefficients.

We assume for purposes of the theorem above that some data values are coarsely pruned in an online manner so that the number of nonzero coefficients stays roughly the same during the course of the decomposition. A similar approach is used effectively in our earlier work [VW99]. Otherwise the number of nonzero coefficients could increase in the worst case by a factor of $\prod_{1\le i\le d}\log|D_i|$. The final (and major) thresholding is done on the coefficients that make it through the initial filter.

4.4 Thresholding

In our method, we compute the wavelet decomposition for the four-dimensional arrays P and V. Given the storage limitation for the histogram, we can only "keep" a certain number of the wavelet coefficients for the p-domain of point counts and the v-domain of vertex counts, which are defined in Section 4.2. Let m denote the number of wavelet coefficients that we have room to keep. Let m_1 and m_2 (where $m_1 + m_2 = m$) denote the number of coefficients that we keep for the p-domain and v-domain, respectively. The goal of thresholding is to determine which are the "best" m_1 and m_2 coefficients to keep, so as to minimize the error of estimating s_{MBR} and c_{cand}.

It is well-known that thresholding by choosing the m_1 largest (in absolute value) normalized wavelet coefficients is provably optimal in terms of minimizing the 2-norm of the absolute error in the p-domain. The corresponding statement is true for the v-domain. If we do not have any prior knowledge of the pattern/distribution of the window queries, we can just choose the top m_1 and m_2 coefficients from the p-domain and v-domain independently. For each coefficient chosen, we need to store the value of the coefficient together with its index (in an appropriate one-dimensional order of the grid cells). Thus the m coefficients may require $2m$ numbers in storage size.

The simplest way to choose the m coefficients is to pick the top $m/2$ coefficients in the two domains independently. Since the p-domain is more important in the sense that it affects the estimation of both s_{MBR} and c_{cand}, we may want to use some heuristic to pick more coefficients from the p-domain. Another heuristic we can use is to pick the coefficients in the two domains so that the indices can be shared. For example, we can pick the top m_1 coefficients from the p-domain first. When picking the m_2 coefficients from the v-domain, we consider both the magnitude of the coefficients and their indices. A coefficient is weighted more if its index appears as an index of one of the already chosen m_1 coefficients. For the pair of coefficients that share the same index, we need only three (rather than four) numbers to store two wavelet coefficients, thus allowing us to increase the effective number m of coefficients that we can keep.

5 Estimating the Cost of Spatial Selections

In this section, we show how to estimate the cost of window queries using the wavelet-based histogram constructed in the previous section in a time-efficient and space-efficient manner.

Let us consider the p-domain first. The naive way of estimating the MBR selectivity for a given window query is to perform wavelet reconstruction based upon the m_1 wavelet coefficients we obtained during the offline stage, with all the other coefficients being set to 0. The reconstruction is an approximate representation of the P array. We can then use formula (6) to compute s_{MBR}. However, the space and CPU complexity of doing the normal wavelet reconstruction are both $O(G)$. That is, even we only keep $m \ll G$ coefficients in the histogram, we need $O(G)$ space and CPU time to use the histogram to do the cost estimation during the online phase! This approach is certainly too costly to be acceptable.

Using the properties of wavelet decomposition, we can design efficient online estimation algorithm. Based upon the *error tree* structure introduced in [MVW98] and the lemmas derived in Section 6 of [VW99], we use the following algorithm to compute s_{MBR} of query q whose MBR can be (approximately) represented as (q_1, q_2, q_3, q_4) in the grid space.

Let $\mathit{coeff}[1, \ldots, m_1]$ denote the m_1 coefficients we obtained by wavelet decomposition of the p-domain array P.

```
$\mathit{CostEstimation}(\mathit{coeff}, m_1, q_1, q_2, q_3, q_4)$
    $s_{MBR} = 0$;
    for $i = 1, 2, \ldots, m_1$ do
        if $\mathit{Contribute}(\mathit{coeff}[i], q_2, q_1, q_4, q_3)$
            $s_{MBR} = s_{MBR} + \mathit{Compute\_Contribution}(\mathit{coeff}[i], q_2, q_1, q_4, q_3)$;
    return $\mathit{answer}$;
```

Function $\mathit{Contribute}(\mathit{coeff}[i], i_1, i_2, i_3, i_4)$ returns *true* if $\mathit{coeff}[i]$ contributes to the the reconstruction of value $P(i_1, i_2, i_3, i_4)$, and it returns *false* otherwise. The actual contribution of $\mathit{coeff}[i]$ to the specified value is computed by function $\mathit{Compute\_Contribution}(\mathit{coeff}[i], i_1, i_2, i_3, i_4)$.

Using the regular structure of the error tree and the lemmas in [VW99], we have devised two algorithms to compute these two functions. Each algorithm has constant CPU time complexity. For reasons of brevity, we refer to [Wan99] for the details. The complexity c_{cand} of a window query can be computed in a similar way.

Theorem 2 *For a given window query, the approximate MBR selectivity s_{MBR} and complexity c_{cand} can be computed based upon the $m = m_1 + m_2$ coefficients in the histogram using a $(2m)$-space data structure in $\min\{m, \prod_{1 \le i \le 4} \log |D_i|\}$ CPU time.*

Proof Sketch: Each of the coefficients can be stored using two numbers, one for its cell number and one for its value. So the space complexity is at most $2m$. (Typically the space complexity is less since the indices can be shared, as mentioned at the end of Section 4.4.) The CPU time complexity follows easily from that of the two functions. An alternate mechanism is to process only the coefficients needed, which are at most $\prod_{1 \le i \le 4} \log |D_i|$. □

It is very important to note that our algorithm has the useful feature that it can progressively refine the approximate answer with no added overhead. If a coefficient contributes to a query, its contribution can be computed independently of the other coefficients. Therefore, to refine a query answer, the contribution of a new coefficient can be added to the previous answer in constant CPU time, without starting over from scratch.

6 Experiments

In this section, we describe the experiments that we performed to evaluate the performance of our proposed wavelet method for spatial selectivity. As a reference, we also include the results for the SQ-histogram method of Aboulnaga and Naughton [AN00]. We first describe our error metric, the data, and the query generation procedures. We also explain some of the parameters to the SQ-histogram and wavelet methods, because the constraints on them may not be immediately apparent. We will then discuss our experiments and the performance of the different methods under varying query size, memory, selectivity and data set aspect ratio.

6.1 Error Metric

Our measure of performance is the average relative error and average absolute error of the estimated MBR selectivity and the estimated complexity of the candidate set. For brevity we report only the relative errors in this section. The results are even better in terms of absolute error. The quantities that we measure are actually not s_{MBR} and c_{cand} but rather $N \times s_{MBR}$ (the size of the candidate set) and $N \times s_{MBR} \times c_{cand}$ (the total number of vertices of the objects in the candidate set).

Average relative error is defined as

$$\textit{Ave. Relative Error} = \sum_{\text{queries}} \frac{|x_{\text{est}} - x_{\text{real}}|}{x_{\text{real}}} \tag{8}$$

where x_{est} is the estimated value and x_{real} is the actual measured value.

6.2 Parameters

In the SQ-histogram, three parameters affect the quality of the constructed quadtrees: the number t of trees, the maximum number ℓ of levels of a tree, and the number of bytes M of allocated memory. The number of levels affects the granularity in the p-domain, and the number of trees affects the granularity in the v-domain. By construction each ℓ-level tree requires at least ℓ buckets and therefore we must have $t \times \ell \times \textit{bucketsize} \leq M$, where *bucketsize* is the number of bytes needed to store all the summary information for each bucket of the SQ-histogram.

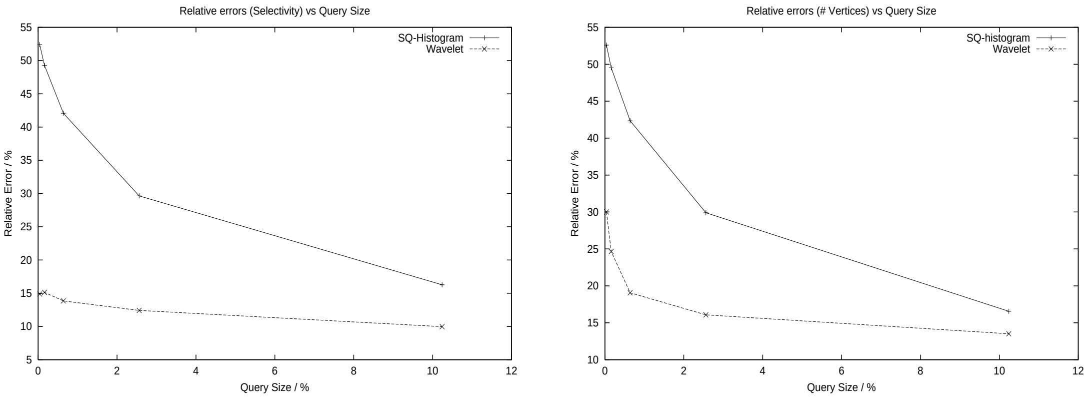
In the wavelet method, we assume sharing of indices between the coefficients of the p-domain and the v-domain. Let m_1 and m_2 be the number of coefficients used in the p-domain and v-domain respectively. If $m_1 \geq m_2$ and if we allocate 4 bytes for each index and each coefficient, then $4(2m_1 + m_2) \leq M$. We generally want $m_1 \geq m_2$ because c_{cand} is dependent on s_{MBR} and it is reasonable to allocate more coefficients to m_1. In our experiments, we set $m_1 \approx 2m_2$.

6.3 Data Set

For our experiments we use the same data generator developed by Aboulnaga and Naughton [AN00] for their experiments on SQ-histograms. In this paper we report experiments on a representative data set consisting of 10,000 rectangles in a $16,384 \times 16,384$ bounding box. Each rectangle represents the MBR of a polygon and is associated with a number representing the number of vertices of that polygon. This number is randomly chosen from the interval $[3, 1000]$. Each rectangle is generated by first randomly putting it into one of four bins. Three of the bins each correspond to a cluster, while the last one corresponds to the unclustered rectangles. Each cluster is associated with a cluster MBR and an area interval. We randomly pick a point in the cluster MBR as the center of the rectangle and a value in the area interval as the area of the rectangle. After randomly picking an aspect ratio in the range $[1, 8]$, we can determine the coordinates of the rectangle. The last step is to randomly flip the rectangle so that the longer sides will not always be along one axis.

6.4 Query Workloads

In our experiments we use five query workloads with 10000 query windows each. Each query workload has a fixed average query size ranging from 0.04% to 10.24% of the data set MBR area. The query windows are generated by randomly choosing a center from the centers of the rectangles in the data set, an aspect ratio from the range $[1, 8]$, and an area value uniformly distributed in the interval $[95\% \times \textit{qsize}, 105\% \times \textit{qsize}]$, where *qsize* is the desired average query size.



(a) Selectivity

(b) No. of Vertices

Figure 3: Performance of the various methods under varying query window size.

6.5 Varying Query Size

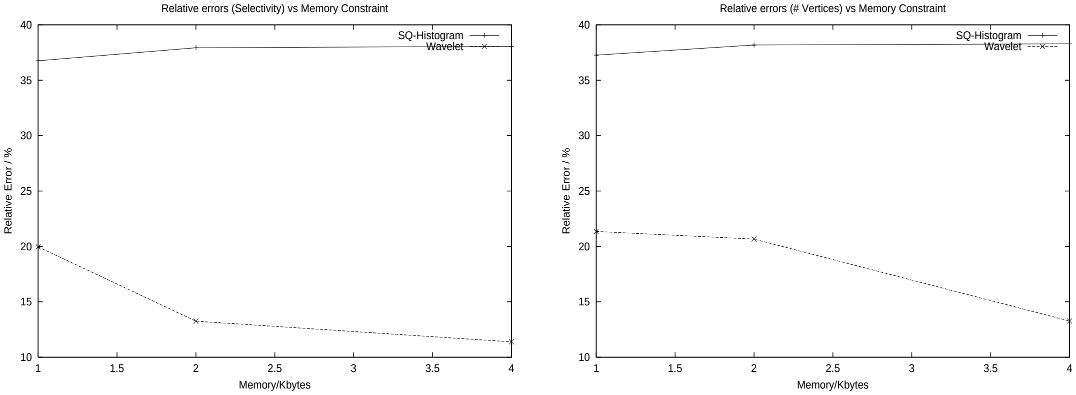
In this experiment we tested the different methods on query workloads of five different average query sizes: 0.04%, 0.16%, 0.64%, 2.56%, and 10.24%. The amount of memory each method can use is fixed at 2048 bytes. For the SQ-histogram, we set the maximum number of levels to 6 and the number of trees to 10. These values were chosen so that the trees will fit into the 2048 bytes of memory. For the wavelet methods, we used 64 grid divisions per axis, 206 coefficients for the p-domain and 100 coefficients for the v-domain.

Figure 3 shows the average relative errors of the different methods. We note that the relative error for the wavelet method is much smaller than that of the SQ-histogram in both the p-domain and the v-domain, especially when the query size is less than 10% of the entire area, which is the case important in practice.

6.6 Varying Memory Constraint

Another question we are interested in is how do these methods scale with memory size allocated. We apply the different methods on the same five query workloads using 1024 bytes, 2048 bytes, and 4096 bytes of memory. The maximum number of levels used in the SQ-histogram method and the number of coefficients used in the wavelet method are summarized in Table 1. The number of trees used in the SQ-histogram method is fixed at 10 for all runs.

The average relative error is computed for each memory size over all five query workloads. Figure 4 shows the results. In both the p-domain and the v-domain, not only is the wavelet method more accurate, but its accuracy improves with more memory. Note for the SQ-histogram that its accuracy does not necessarily improve with more memory, because the size of the quadtrees after merging is also dependent on the size and location of the rectangles in the data set. This situation arises, for example, when either the height or width of each MBR is at least one-half of that of the input space. In that case, all the MBRs will be stored



(a) Selectivity

(b) No. of Vertices

Figure 4: Performance of the various methods under varying memory constraints.

in the root node; the resulting histogram will consist of a single bucket, regardless of the memory available.

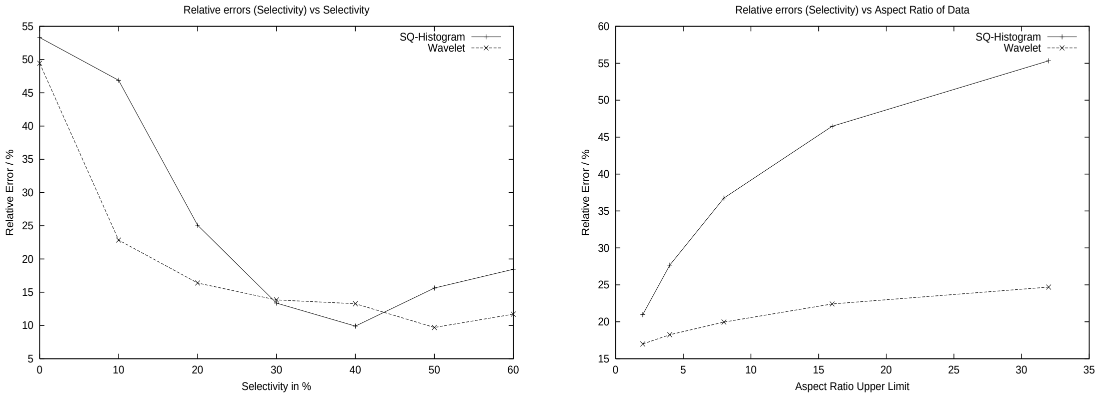
6.7 Varying Query Window Selectivity

In this section, we look at our results from an alternative view. We partition the results (with 1024 bytes memory constraint) according to the actual selectivity of each query window. The partitions we used are $\{[0,10), [10,20), [20,30), [30,40), [40,50), [50,60), [60,70), [70,80), [80,90), [90,100]\}$, where the values are percentages of the total number of objects in the data set. The partitions are indexed by their lower endpoint, so 0% on the plot corresponds to the interval $[0,10)$.

Figure 5(a) shows the repartitioned results. It turns out that all of our query windows overlap with 0% to 70% of the data set. The figure shows that the wavelet method is more accurate than SQ-histogram over this range of query window selectivity most of the time.

Memory (bytes)	SQ-histogram	Wavelet	
M	ℓ	m_1	m_2
1024	3 levels	103	50
2048	6 levels	206	100
4096	8 levels	412	200

Table 1: Parameters used in the SQ-histogram method and the wavelet method. The numbers in the wavelet columns are the number of coefficients used for the p-domain and the v-domain respectively.



(a) Varying query window selectivity.

(b) Varying data set aspect ratio.

Figure 5: Performance of the various methods in estimating selectivity under varying query window selectivity and varying data set aspect ratio.

6.8 Varying Data Set Aspect Ratio

We also studied how the different methods perform on data of different aspect ratios. We generated five data sets whose objects have randomly chosen aspect ratios from different ranges $\{[1,2],[1,4],[1,8],[1,16],[1,32]\}$. Each of these data sets have 10,000 rectangles in a $16,384 \times 16,384$ space as before. The performance of the different methods are shown in Figure 5(b). Our wavelet method is significantly more stable and robust under increasing aspect ratio then the other methods. Long and skinny horizontal (or vertical) rectangles of different sizes and locations will typically be placed near the root level of the quadtrees in the SQ-histogram, rendering the objects within a node or a bucket less uniform.

7 Conclusions

Cost estimation in spatial databases is a hard problem in query optimization because of the complex features of the spatial data and queries. In this paper, we present a new method for spatial cost estimation that improves over partition-based histogram techniques. By transforming the spatial objects in object space into higher-dimensional points in a transformed space, we can conveniently capture several features of the spatial objects. The point data distribution in object space is then effectively approximated through a novel data compression technique based on a wavelet decomposition of partial sums in the transformed space. We overcome the major problem that the partial sum data are very dense and show how to indirectly process the data via sparse techniques to compute the wavelet decomposition in a space-efficient way. Experiments confirm that our method is robust over a broad range of spatial data distributions and queries and can provide accurate cost estimations even with very small space allocation. We do not give analytic guarantees of accuracy or confidence bounds, but performance is uniformly good.

Besides accuracy, another important problem for cost estimation is how to maintain the histogram when the underlying data distribution changes. In [MVW00], we introduce an efficient method for the dynamic maintenance of wavelet-based histograms. An interesting and challenging extension of those techniques would be to the partial sum arrays we consider in this paper.

Acknowledgments. We gratefully acknowledge earlier discussions with Ramesh Agarwal, Bishwaranjan Bhattacharjee, Ming-Ling Lo, and Tim Malkemus. We especially thank Ashraf Aboulnaga for providing us the source code of his synthetic data generation and SQ-histogram construction programs.

References

[AN00] A. Aboulnaga and J. F. Naughton. Accurate estimation of the cost of spatial selections. In *Proceedings IEEE International Conference on Data Engineering*, San Diego, California, 2000.

[APR99] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity estimation in spatial databases. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 13–24, Phildelphia, June 1999.

[BF95] A. Belussi and C. Faloutsos. Estimating the selectivity of spatial queries using the 'correlation' fractal dimension. In *Proceedings of 21th International Conference on Very Large Data Bases*, pages 299–310, Zurich, Switzerland, September 1995.

[FK94] Christos Faloutsos and Ibrahim Kamel. Beyond uniformity and independence: Analysis of r-trees using the concept of fractal dimension. In *Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 4–13, Minneapolis, Minnesota, May 1994. ACM Press.

[KF93] Ibrahim Kamel and Christos Faloutsos. On packing r-trees. In Bharat K. Bhargava, Timothy W. Finin, and Yelena Yesha, editors, *CIKM 93, Proceedings of the Second International Conference on Information and Knowledge Management, Washington, DC, USA, November 1-5, 1993*, pages 490–499. ACM, 1993.

[MD88] M. Muralikrishna and D. J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, pages 28–36, 1988.

[MVW98] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 448–459, Seattle, WA, June 1998.

[MVW00] Y. Matias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *Proceedings of the 2000 International Conference on Very Large Databases*, Cairo, Egypt, September 2000.

[NH85] J. Nievergelt and K. Hinrichs. Storage and access structures for geometric data bases. In *Proceedings of International Conference on Foundation of Data Organization*, pages 335–345, 1985.

[Ore86] Jack A. Orenstein. Spatial query processing in an object-oriented database system. In Carlo Zaniolo, editor, *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data*, pages 326–336, Washington, D.C., May 1986.

[PI97] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proceedings of the 1997 International Conference on Very Large Databases*, Athens, Greece, August 1997.

[PIHS96] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, May 1996.

[PSC84] G. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 256–276, 1984.

[PSTW93] Bernd-Uwe Pagel, Hans-Werner Six, Heinrich Toben, and Peter Widmayer. Towards an analysis of range query performance in spatial data structures. In *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 25-28, 1993, Washington, DC*, pages 214–221. ACM Press, 1993.

[SK88] B. Seeger and H. P. Kriegal. Techniques for design and implementation of efficient spatial access methods. In *Proceedings of 14th International Conference on Very Large Data Bases*, pages 360–371, 1988.

[TS96] Y. Theodoridis and T. K. Sellis. A model for the prediction of r-tree performance. In *Proceedings of the Fifteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 161–171, Montreal, Canada, June 1996.

[Vit99] J. S. Vitter. External memory algorithms and data structures. In J. Abello and J. S. Vitter, editors, *External Memory Algorithms and Visualization*. American Mathematical Society Press, Providence, RI, 1999.

[VW99] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pages 193–204, Phildelphia, June 1999.

[VWI98] J. S. Vitter, M. Wang, and B. Iyer. Data cube approximation and histograms via wavelets. In *Proceedings of Seventh International Conference on Information and Knowledge Management*, pages 96–104, Washington D.C., November 1998.

[Wan99] M. Wang. *Approximation and Learning Techniques in Database Systems.* Ph. D. dissertation, Duke University, 1999. The thesis is available via the author's web page `http://www.cs.duke.edu/~minw/`.

[WKW94] Kyu-Young Whang, Sang-Wook Kim, and Gio Wiederhold. Dynamic maintenance of data distribution for selectivity estimation. *VLDB Journal*, 3(1):29–51, 1994.