Selectivity Estimation of Complex Spatial Queries

Nikos Mamoulis[1] and Dimitris Papadias[2]

[1] CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands
nikos@cwi.nl
[2] Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong
dimitris@cs.ust.hk

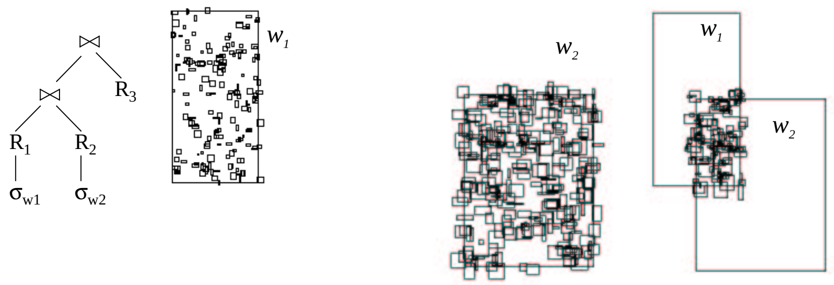
Abstract. Several studies have focused on the efficient processing of simple spatial query types such as selections and spatial joins. Little work, however, has been done towards the optimization of queries that process several spatial inputs and combine them through join and selection conditions. This paper identifies the dependencies between spatial operators and illustrates how they can affect the outcome of complex queries. A thorough analysis yields selectivity estimations that can be used to optimize any combination of spatial and non-spatial selection and join operators. The accuracy of the formulae is evaluated through experimentation with various queries. In addition to their importance for spatial databases, the presented results can be applied in several other domains, where dependencies exist between operators.

1. Introduction

Relational queries are processed by combining operators (e.g., sort-merge, hash-join, index-based search) in an optimal way [10]. Optimization algorithms (e.g., dynamic programming) search through the space of valid plans in order to identify one with low cost. Most query optimizers estimate the output sizes using catalog information (e.g., size and distribution) about the relations involved in the queries. The core assumption is that there are no dependencies between different attributes. Therefore, the results of any complex query are assumed to depend solely on the distribution of the data from the base tables. In many cases the independence assumption holds and the optimizer returns accurate results. For instance, assume that two tables Employee and Department are joined through the common attribute DeptId, and the following selections apply: Employee.Age>35 and Department.Sales<10000. The DeptId values after the application of selections are expected to maintain their initial distribution, since there is no explicit dependency between the query attributes. Thus the cost of the join can be estimated using the selectivity of the selections and statistical information about the base tables.

In spatial database systems [9], a complex query may contain several spatial and non-spatial components. For instance, the query "find all cities within 400km of Munich, which are less than 10km away from a forest and are crossed by a river wider than 20m” includes two spatial joins (City $\bowtie$ Forest, City $\bowtie$ River), a non-spatial selection (River.Width > 20m) and a spatial selection (City.CRegion *within 400km of* Munich). Although there may be no dependency between the non-spatial attribute River.Width with the spatial (i.e., location/extent), notice that there is *always* a dependency between the spatial operators involved in a query. We do not expect that cities located in America or Asia will be part of the result. Therefore the spatial selections affect not only the number of objects that will participate in a succeeding join, but also their spatial distribution, which determines the query result.

The next example strengthens our point. Consider the plan of Figure 1a, where R_1, R_2 are spatial relations, σ_{w1}, σ_{w2} spatial selections (e.g., window queries) and R_3 a third, not necessarily spatial, relation. Figures 1b and 1c illustrate example results of the selections applied to the corresponding dataset. Clearly the cost of the last join depends on the selectivity of the first (spatial) one. The output of the spatial join, however, does not depend solely on the results of the window queries, but also on their relative position. As Figure 1d shows, the output size of the join increases with the intersection area of the windows since only objects inside or near the intersection may participate in the result. If the windows are far apart, the result of the spatial join is expected to be empty.



(a) a query plan (b) $\sigma_{w_1}(R_1)$ (c) $\sigma_{w_2}(R_2)$ (d) $\sigma_{w_1}(R_1) \bowtie \sigma_{w_2}(R_2)$

Fig. 1. Example of spatial operator dependency

Although interdependencies between selection and join attributes are not common in non-spatial queries (i.e., relations are typically joined on a key attribute, whereas selections apply on non-keys), they always exist in spatial queries because there is typically only one spatial attribute per relation. This paper studies selectivity estimation of complex spatial queries that involve spatial selections and joins. More specifically, given a query consisting of *n* relations joined on their spatial attribute and potentially restricted by selection windows, we provide accurate formulae that estimate its output size. Following the common conventions in spatial query optimization, we assume that the input data are uniformly distributed and that the predicate is *intersect* (*overlap*). The proposed formulae use catalog information about the mean sizes of the objects in spatial relations to estimate the query result.

Our techniques can be applied for arbitrarily distributed datasets using local statistics like 2D-histograms. They are also appropriate for spatial queries with predicates other than *intersect* (e.g. *meets*, *covers*), since such queries can be transformed to intersection queries (see [20] for a case study). Even more importantly, the methods are not strictly limited to the spatial domain, because dependencies between operators can also be found in other database applications. The rest of the paper is organized as follows. Section 2 provides background on processing and optimization of simple spatial query types and multiway spatial joins. Section 3 presents formulae that estimate the selectivity of complex spatial queries. In Section 4 we evaluate the accuracy of the proposed models for complex spatial queries on uniform data. Section 5 discusses extensions to real data and Section 6 concludes the paper.

2. Background

Spatial database systems [9] organize and manage large collections of multidimensional data. Spatial relations, apart from conventional attributes, contain one attribute that captures the geometric features of the stored objects. For example, the last attribute in relation City(*CName, PostalCode, Population, CRegion*) is of spatial type *polygon*. In addition to traditional data structures (e.g., B-trees) for alphanumeric attributes, spatial relations are indexed by *multidimensional access methods* [7], usually R-trees [8], for the efficient processing of queries such as *spatial selections* (or window queries) and *spatial joins*. Selections (e.g., "cities in Germany") apply on a single relation, while spatial joins (e.g. "cities *crossed by* a river") combine two relations with respect to a spatial predicate (typically *intersect*, which is the counterpart of the relational equi-join).

Complex spatial queries include spatial and non-spatial selections and joins. They can be processed by combining simple operators in a processing tree (plan) like the one illustrated in Figure 1a. The efficiency of an operator depends on whether its input (inputs) is (are) indexed. For instance, the cost of a window query applied on an R-tree is typically linearly related to the size of the window. On the other hand, if the selection applies on intermediate results, the whole input needs to be scanned independently of the selectivity of the operator. The same applies for join operators. The most efficient spatial join method is the R-tree join (RJ) [5], which matches two R-trees. Some methods [15, 18] join an R-tree with some non-indexed dataset (e.g., an intermediate result of another operator). Others [16, 19, 14, 2] organize two non-indexed inputs in intermediate file structures (e.g., hash buckets) in order to join them efficiently in memory.

Typically, a complex query has a large number of potential execution plans with significant cost differences. For instance, an alternative plan to that of Figure 1a would be to first join R_2 with R_3, then apply the selection on R_2 and finally join the intermediate result with R_1, after it has been restricted by w_1. The number of plans increases exponentially with the number of involved relations (see [18, 25] for an analysis on spatial and non-spatial domains). Optimization algorithms search either in a deterministic (e.g., dynamic programming) or a randomized way (e.g., hill climbing [12]) to find a cheap plan. The cost of a specific plan is computed using formulae for (i) the operators involved in the plan, and (ii) the output size of each sub-query of the plan. The first provide an estimate for the cost of each node in the plan, while the second determine the cost of succeeding operators. In the query of Figure 1a, for instance, the selectivity of the spatial join $R_1 \bowtie R_2$ affects the cost of the final operator.

There has been extensive research on the accurate estimation of the selectivity and cost of spatial operators. The selectivity of spatial selections has been studied as a prerequisite for the I/O cost estimation of R-tree window queries [13, 24, 23, 27]. Given a spatial dataset R of N d-dimensional uniformly distributed rectangles in a rectangular area r (workspace), the number of rectangles that intersect a window query w (*output cardinality - OC*) is estimated by the following formula:

$$OC(R,w) = N \cdot \prod_{d=1}^{k} \min\left(1, \frac{\overline{s_d} + \overline{w_d}}{\overline{r_d}}\right) \quad \textbf{(1)}$$

where $\overline{s_d}$ is the average length of the projection of a rectangle $s \in R$ at dimension d. $\overline{w_d}$ and $\overline{r_d}$ are the corresponding projections of w, r respectively. The last factor (product) of Equation 1, called Minkowski sum, is the *selectivity* of the window query (i.e., the probability that a random rectangle from R intersects w). This probability at some dimension d equals the sum of projections $\overline{s_d}$ and $\overline{w_d}$ on that dimension normalized to the workspace. Equation 1 can be extended for the output cardinality of an (intersection) spatial join between two relations R_1 and R_2 as follows [28]:

$$OC(R_1,R_2) = N_1 \cdot N_2 \cdot \prod_{d=1}^{k} \min\left(1, \frac{\overline{s_{1,d}} + \overline{s_{2,d}}}{\overline{r_d}}\right) \quad \textbf{(2)}$$

N_1, N_2 denote the cardinalities of the datasets, and $\overline{s_{1,d}}$, $\overline{s_{2,d}}$ correspond to the average length of the projection of rectangles $s_1 \in R_1$ and $s_2 \in R_2$ on dimension d. In other words, the expected number of rectangle pairs that intersect is equal to the number of results after applying N_2 window queries of area s_2 on R_1.

The selectivity of multiway spatial joins can be accurately estimated only for acyclic and clique (i.e., complete) query graphs. Let R_1, ..., R_n be n spatial datasets joined through a query graph Q, where Q_{ij} = True, iff rectangle $r_i \in R_i$ should intersect rectangle $r_j \in R_j$. When Q is acyclic, the number of qualifying object combinations can be estimated by:

$$OC(R_1,...,R_n,Q) = \prod_{i=1}^{n} N_i \cdot \prod_{\forall i,j:Q_{ij}=TRUE} \prod_{d=1}^{k} \min\left(1, \frac{\overline{s_{i,d}} + \overline{s_{j,d}}}{\overline{r_d}}\right) \quad \textbf{(3)}$$

The above formula actually restricts the Cartesian product of the datasets using the selectivities of the query edges, which are independent. When the query graph contains cycles, the pairwise selectivities are no longer independent. For instance, if a intersects b and b intersects c, the probability that a intersects c is relatively high because the rectangles are expected to be close to each other. Thus Equation 3 is not appropriate for such cases. For the special case where Q is complete (clique) a closed formula that estimates the output of the multiway join is proposed in [21]:

$$OC(R_1,...,R_n,Q) = \prod_{i=1}^{n} N_i \cdot \prod_{d=1}^{k} \frac{1}{(n-1)\cdot \overline{r_d}} \sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} \overline{s_{j,d}} \tag{4}$$

This formula can be derived by the observation that if $\{s_1, s_2, \ldots, s_{n-1}\}$ is a clique then $\{s_1, s_2, \ldots, s_{n-1}, s_n\}$ is also a clique iff s_n intersects the common area of all rectangles in $\{s_1, s_2, \ldots, s_{n-1}\}$. Equations 1 through 4 are accurate for uniform datasets covering the same workspace r. In real life applications these assumptions may not hold, therefore researchers have extended some of them for skewed datasets. A histogram-based method that estimates the selectivity of window queries is presented in [3]. This method decomposes the space irregularly using buckets that cover objects of similar density and keeps statistical information for each bucket, considering that its contents are uniform. A similar technique that divides the objects using a quad-tree like partitioning is presented in [1]. Another method that applies only on point datasets and uses theoretical laws is proposed in [4]. Regarding the selectivity of spatial joins, relatively little work has been done. In [28, 18] the space is decomposed using a regular grid and uniformity is assumed for each cell. The output of the join is then estimated by summing up the estimations from each cell. In [6] an interesting law that governs the selectivity of distance spatial joins (i.e., is joins that return point pairs within a distance parameter) is presented.

As discussed in the introduction, the relative positions of selection windows determine the skew of the joined rectangles from each dataset. Thus existing formula that focus exclusively on spatial selections or joins are not applicable. In the next section we study the selectivity of complex spatial queries, where the dependency of operators affects the query result.

3. Selectivity of complex spatial queries

When only one selection window w_i that applies on a single dataset R_i exists, selectivity estimation is rather simple. Since the result is the same independently of the order according to which operators are applied, we can assume that the selection follows the join. The output cardinality can then be estimated by multiplying the corresponding join formulae with the selectivity of the window query:

$$OC(Join, w_i) = OC(Join) \cdot \prod_{d=1}^{k} \min\left(1, \frac{\overline{s_d} + \overline{w_d}}{\overline{r_d}}\right) \tag{5}$$

where $OC(Join)$ can be any of Equations 2, 3 or 4.

The problem is more complicated when two or more spatial selections exist on the joined datasets. A simple approach is to assume that the join workspace is the common area of all selections, and apply a single selection on the multiway join result using this area. This, however, would be inaccurate since the common area of selection windows may be empty, while there may exist objects that qualify the query, especially if the non-intersecting windows are close to each other and the data rectangles are large.

3.1 Selectivity of a pairwise join restricted by two selections

Let us first confine our study to the special case where a pair of joined datasets R_1 and R_2 are restricted by two query windows w_1 and w_2, respectively. Based on the extents of w_1 and w_2 we will try to identify the area that should be intersected by rectangles from each dataset in order to participate in the join. Thus, we will compute the query selectivity in three steps: (i) determine a number of candidate join objects from each dataset using w_1, w_2, (ii) estimate the workspace area of the join and (iii) compute the join selectivity using the number of candidates, the estimated workspace area and the average rectangle extents according to Equation 2.

Let $w_{i,d}$ be the projection of w_i on dimension d, $w_{i,d,s}$ and $w_{i,d,e}$ be the starting and the ending point of $w_{i,d}$, respectively, and $\overline{w_{i,d}}$ be the length of $w_{i,d}$. Consider also similar notations for the corresponding properties of the average extents of a rectangle s_i in dataset R_i (e.g., $\overline{s_{i,d}}$ is the average projection of s_i on dimension d). We define two *updated* windows w_1', w_2', as follows:

$$w_{1,d,s}' = \max\{w_{1,d,s}, w_{2,d,s} - \overline{s_{2,d}}\} \quad \textbf{(6a)}$$

$$w_{1,d,e}' = \min\{w_{1,d,e}, w_{2,d,e} + \overline{s_{2,d}}\} \quad \textbf{(6b)}$$

$$w_{2,d,s}' = \max\{w_{2,d,s}, w_{1,d,s} - \overline{s_{1,d}}\} \quad \textbf{(6c)}$$

$$w_{2,d,e}' = \min\{w_{2,d,e}, w_{1,d,e} + \overline{s_{1,d}}\} \quad \textbf{(6d)}$$

In order for a rectangle from R_i to be candidate for the join *and* intersect w_i, it should intersect w_i'. For instance, consider the selection windows w_1, w_2 and the updated ones w_1', w_2' in Figure 2a. Object *a*, belonging to R_1 and intersecting w_1, cannot participate in the join because it may not overlap some object from R_2 that intersects w_2. On the other hand, object *b* that intersects w_1' is a potential query result.

Intuitively, w_1' and w_2' define the area where the spatial join is restricted. The number of rectangles that participate in the join from dataset R_i is determined by the selectivity of the corresponding w_i'. Notice that the end points set by Equations 6a-d do not always define valid intervals, since the lower end point may be greater than the upper end point. This situation may arise when the actual windows do not intersect and there is a large distance between them. In this case (if $w_{i,d,s}' > w_{i,d,e}'$ for some i, d), the length of the corresponding projection is negative, and the selectivity of $w_{i,d}'$ is positive only when $\overline{s_{i,d}} > w_{i,d,s}' - w_{i,d,e}'$; otherwise, the selectivity of the complex query is zero.

So far, we have calculated the windows w_1' and w_2' that should be intersected by each rectangle from R_1 and R_2, respectively, in the result. These windows can be used in combination with Equation 1 to determine the number of join candidates from each dataset. The next step is to estimate the *workspace of the join c* , i.e., the area where the query results lie. The rectangles from R_i that may participate in the join are *inside* a window c_i, which is generated by extending $w_{i,d}'$ with $\overline{s_{i,d}}$ at both sides on each dimension. For instance, in Figure 2b we know that join candidates from R_1 are included in c_1. Since c_1 and c_2 do not cover the same area, we need to average them in order to acquire a unique rectangle c that reflects best the area where the spatial join is restricted. Figure 2c provides an example of this normalization. Formally:

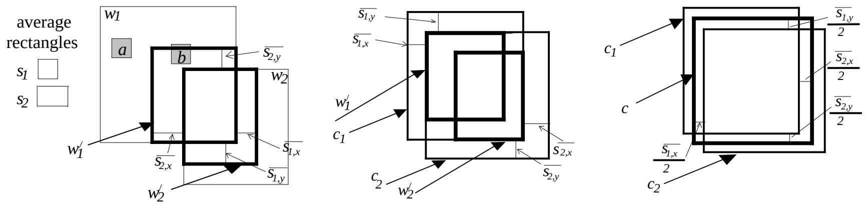
$$c_{d,s} = \max\{w_{1,d,s}' - \overline{s_{1,d}}\ ,\ w_{2,d,s}' - \overline{s_{2,d}}\ \} - |w_{1,d,s}' - \overline{s_{1,d}} - w_{2,d,s}' + \overline{s_{2,d}}\ |/2 \qquad \textbf{(7a)}$$

$$c_{d,e} = \min\{w_{1,d,e}' + \overline{s_{1,d}}\ ,\ w_{2,d,e}' + \overline{s_{2,d}}\ \} + |w_{1,d,e}' + \overline{s_{1,d}} - w_{2,d,e}' - \overline{s_{2,d}}\ |/2 \qquad \textbf{(7b)}$$

Figure 3 shows some more one-dimensional examples with four representative window configurations over one pair of datasets. In case 3, the workspace is non-empty, although the original windows do not intersect. In case 4, the updated windows w_1', w_2' are invalid; a rectangle from R_i should intersect both endpoints of the invalid window w_i' in order to be a candidate join object. However, the average length of a rectangle $s_1 \in R_1$ is smaller than the distance of the endpoints of w_1', thus the join is considered to have zero selectivity.

Given the join workspace c, selectivity can be computed using the existing formulae for spatial selections and joins. Summarizing, the output cardinality of a query that includes the spatial join of two datasets R_1, R_2 restricted by window queries w_1, w_2, respectively, is estimated by the following formula:

$$OC(R_1, R_2, w_1, w_2) = OC(R_1, w_1') \cdot OC(R_2, w_2') \cdot \prod_{d=1}^{k} \min\left(1, \frac{\overline{s_{1,d}} + \overline{s_{2,d}}}{\overline{c_d}}\right) \qquad \textbf{(8)}$$



(a) updated windows w_1', w_2' (b) generation of c_1, c_2 (c) join workspace c

Fig. 2. Generation of join workspace

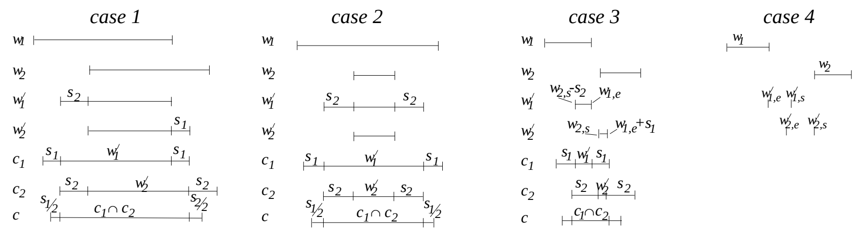


Fig. 3. Windows that define the selectivity of a complex pairwise join

3.2 Selectivity of multiway joins with selections

Equation 8 can be extended for complex spatial queries that join *n* datasets, potentially restricted by spatial selections. Selectivity is again computed in three steps. At the first step the updated window w_i' is calculated for each dataset R_i, using the windows of the neighbors in the join graph *Q*. At the second step, depending on *Q*, the workspace area of each join edge or of the whole graph is computed. Finally, the multiway join selectivity is estimated using (i) the selectivity of w_i' for each R_i, (ii) the workspace area(s) and (iii) Equations 3, 4.

The updated selection window w_i' for each R_i is estimated using the initial windows and the query graph. It turns out that the calculation process is not simple, since the update of a window w_i to w_i' may restrict the already updated window w_j' of a neighbor R_j. This process is demonstrated in the example of Figure 4, where three datasets are joined by a chain query and three windows restrict the joined rectangles. Assume that we attempt to calculate the updated window w_i' for each w_i using the following formulae:

$$w_{i,d,s}' = \max\{w_{i,d,s}, \max\{w_{j,d,s} - \overline{s_{j,d}}\ , Q_{ij} = \text{True}\}\} \qquad \textbf{(9a)}$$

$$w_{i,d,e}' = \min\{w_{i,d,e}, \min\{w_{j,d,e} + \overline{s_{j,d}}\ , Q_{ij} = \text{True}\}\} \qquad \textbf{(9b)}$$

First w_1 is restricted to w_1' using w_2 and s_2 (Figure 4c). Then w_2 is restricted to w_2' using w_1, s_1, w_3 and s_3 (Figure 4d). Observe that the left point end of w_2 has changed, and this change should be propagated to w_1' (Figure 4e), which is shortened on the left side. In general, each change at a window should be propagated to all neighbor windows in the query graph.

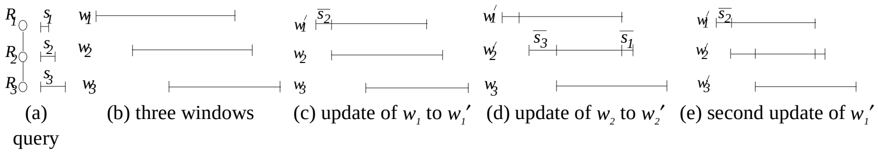


Fig. 4. Selection window update propagation in a network of joined datasets

The problem of window updates is similar to achieving local consistency in constraint satisfaction problems [29]. Therefore, we use a variation of an arc consistency algorithm [17] to estimate the final w_i' for each R_i. The algorithm first places all selection windows in a queue. If a dataset R_i does not have a selection window we set $w_i = r$, i.e., the workspace of the datasets. Then the first window w_f from the queue is picked and updated according to the current windows of the neighbors. If there is a change in w_f, the windows of all neighbors not currently in the queue are inserted in it because the changes need to be propagated to them. The process continues until the queue is empty. The pseudocode of the algorithm is given in Figure 5.

```
window_propagation(window w[], Query Q[][])
 initialize queue;
 for each $R_i$ do
   if $w_i$ does not exist then $w_i$ = r;
   queue.insert($w_i$);
 while not queue.empty() {
   $w_f$ = queue.getfirst();
   for each dimension d do
     $w_{f,d,s}$ = max{$w_{f,d,s}$, max{$w_{j,d,s}$-$\overline{s_{j,d}}$ , $Q_{fj}$ = True}};
     $w_{f,d,e}$ = min{$w_{f,d,e}$, min{$w_{j,d,e}$+$\overline{s_{j,d}}$ , $Q_{fj}$ = True}};
     if $w_f$ has changed
       for each j, $Q_{fj}$ = True do
         if $w_j$ not in queue then queue.insert($w_j$);
 }
```

Fig. 5. The window_propagation algorithm

After the execution of the *window_propagation* algorithm, each window w_i will be transformed to the minimum intersection window w_i'. Window w_i' is determined by the end points of the most restricted window w_j on each dimension. The path connecting R_i with R_j contains at most n-2 graph nodes (where n is the number of datasets involved in the query), meaning that the end points of the window w_i' can be adjusted at most n-1 times per dimension. Thus, the worst case complexity of the algorithm is O($d \cdot n^2$), and its computational overhead in the optimization process is trivial.

The next step of the estimation process is to determine the workspace area c of the multiway join. This process is performed in a similar way as described in the previous paragraph. First the coverage area c_i of each window query w_i' is estimated by extending w_i' with $\overline{s_{i,d}}$ at both sides on each dimension. If the query is acyclic the selectivity of each query edge Q_{ij} is normalized with respect to the corresponding pairwise join workspace. Therefore Equation 3 is modified as follows:

$$OC(R_1,...,R_n,w_1,...,w_n,Q) = \prod_{i=1}^{n} OC(R_i,w_i') \cdot \prod_{\forall i,j:Q_{ij}=TRUE} \prod_{d=1}^{k} \min\left(1, \frac{\overline{s_{i,d}} + \overline{s_{j,d}}}{\overline{\overline{c_{i,j,d}}}}\right) \quad \textbf{(10)}$$

In Equation 10, $c_{i,j}$ denotes the workspace of the pairwise join between R_i and R_j, which is calculated using w_i', w_j', s_i and s_j and Equations 7a, 7b. In case of clique graphs, we need to consider a unique workspace c for the whole multiway join, since all rectangles in an output tuple mutually overlap. This workspace is defined by extending the common intersection i of all workspaces c_i by the average difference of the c_i's from the common intersection at each side and dimension. Formally:

$$i_{d,s} = \max_{1\le i\le n}\{w_{i,d,s}' - \overline{s_{i,d}}\}$$

$$c_{d,s} = i_{d,s} - \sum_{i=1}^{n}\left(i_{d,s} - w'_{i,d,s} + \overline{s_{i,d}}\right)/n \tag{11a}$$

$$i_{d,e} = \min_{1\le i\le n}\{w_{i,d,e}' + \overline{s_{i,d}}\}$$

$$c_{d,e} = i_{d,e} + \sum_{i=1}^{n}\left(w'_{i,d,e} + \overline{s_{i,d}} - i_{d,e}\right)/n \tag{11b}$$

The output cardinality of a multiway clique join is then estimated by the selectivities of the windows and the multiway join selectivity (see Equation 4), normalized to the join workspace c. Formally:

$$OC(Q, R_1,\ldots,R_n, w_1,\ldots,w_n) = \prod_{i=1}^{n} OC(R_i, w_i') \cdot \prod_{d=1}^{k}\frac{1}{(n-1)\cdot \overline{c_d}} \cdot \sum_{i=1}^{n}\prod_{j=1, j\neq i}^{n}\overline{s_{j,d}} \tag{12}$$

4. Experimental Evaluation

In this section we evaluate the accuracy of Equations 8, 10 and 12. For this purpose, we generated four series of synthetic datasets with uniformly distributed rectangles in the square workspace $[0,1)^2$. The density[1] of the datasets in the different series is 0.1, 0.2, 0.4 and 0.8. Each dataset consists of 10,000 rectangles. By U*xy* we will denote the *y*th dataset in the series of density *x*. For instance, U0.1a denotes the first dataset in the series having density 0.1. The lengths of the rectangles are uniformly distributed between 0 and $2\cdot\overline{s_{i,d}}$, where $\overline{s_{i,d}}$ is the rectangle side that leads to the desired density. For instance, in order for a dataset of 10,000 rectangles to have density 0.1, the average rectangle side should be $\sqrt{0.1/10000}$.

Table 1 shows analytical and experimental results on complex pairwise spatial joins. Each row corresponds to a different pair of datasets and each column to a representative configuration of selection windows. Clearly, the estimated output is very close to the actual one. If we define the quantity |estimated-real|/min{estimated, real}

[1] The density of a dataset is defined as the total area of the objects in it divided by the area of the workspace, or else as the expected number of objects that intersect a random point in the workspace.

as estimation error, the median estimation error in the experiment is 8%. The overestimated and underestimated cases are balanced, indicating that the reasoning behind Equation 8 is correct.

Table 1. Evaluation of the estimation formula for pairwise spatial joins with selections

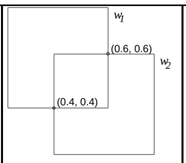
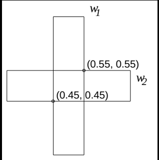
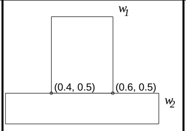
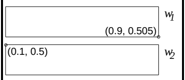
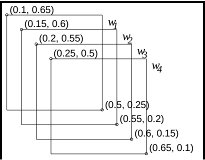
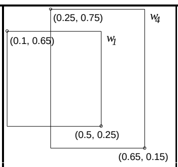
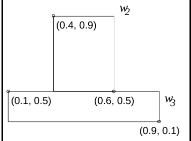
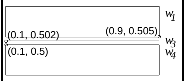
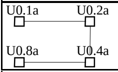
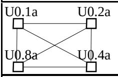
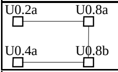
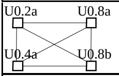
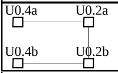
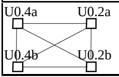
								
	estimated	actual	estimated	actual	estimated	actual	estimated	actual
U0.1a ⋈ U0.8a	633	659	167	179	24	19	46	31
U0.2a ⋈ U0.4a	511	506	134	106	18	12	30	35
U0.1a ⋈ U0.4a	395	406	103	80	12	15	17	17
U0.2a ⋈ U0.8a	780	855	207	252	33	36	69	67
U0.1a ⋈ U0.1b	169	175	43	45	3	3	1	1
U0.8a ⋈ U0.8b	1420	1469	384	433	81	81	199	179

Table 2. Evaluation of the estimation formulae for multiway spatial joins with selections

	 (0.1, 0.65) (0.15, 0.6) w_1 (0.2, 0.55) w_2 (0.25, 0.5) w_3 w_4 (0.5, 0.25) (0.55, 0.2) (0.6, 0.15) (0.65, 0.1)		 (0.25, 0.75) w_4 (0.1, 0.65) w_1 (0.5, 0.25) (0.65, 0.15)					
	estimated	actual	estimated	actual	estimated	actual	estimated	actual
	1136	1107	1784	2464	52	25	40	86
	279	395	443	542	3	3	5	6
	9352	15577	14734	22604	355	425	438	480
	1203	1602	1931	2399	32	54	18	21
	1611	2506	2554	4168	44	17	51	31
	251	348	404	555	4	5	2	3

In the next experiment we test the accuracy of Equations 10 and 12 for multiway spatial joins restricted by selections. We use the uniform datasets described above and several window configurations for chain and clique graphs. Table 2 shows graphically four configurations of windows applied to six join graphs. The assignment of windows to graph nodes is done clockwise, e.g., for the first row w_1 applies to U0.1a, w_2 to

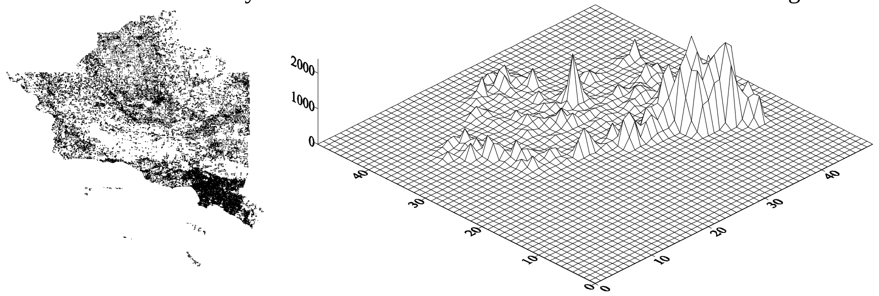
U0.2a, w_3 to U0.4a and w_4 to U0.8a. We have experimented with queries that have windows on all datasets (e.g., first column) and queries with windows on some datasets. Typically the estimation is close to the actual result, but the accuracy is not as high as in the case of binary joins (the median error is now 38%). This happens because of (i) the propagation of the error in partial results and (ii) the fact that the intermediate results are more skewed than the input data. However, this is an unavoidable problem, which also exists in query optimization of relational queries involving many operators [11].

In general, the experiments prove the accuracy of Equations 8, 10 and 12 and support their use for query optimization. However, the importance of this analysis is not only restricted to this task. After proper modification the proposed methods can be used to assure that a query has zero results and, thus avert processing. As explained, if some updated window w_i' is invalid ($w_{i,d,s}' > w_{i,d,e}'$ at some d) the average size of the projection $\overline{s_{i,d}}$ at this dimension determines whether the query *is expected to have any solutions*. If in the above methodology, instead of the average projections we employ the maximum projections max($\overline{s_{i,d}}$) for each dataset on every axis, we can determine whether the query *definitely has no solution*. Thus if $w_{i,d,s}' - w_{i,d,e}' > \max(\overline{s_{i,d}})$ and max($\overline{s_{j,d}}$) is used to derive w_i', processing can be avoided.

5. Extension to real data

In real life applications data are not usually uniform, but the distribution and size of the objects may vary in different areas of the workspace. In such cases, we need models that take advantage of information about the distribution of the objects to estimate the cost of complex queries. Models for range queries and distance joins [4, 6] on point datasets are not readily applicable for intersection joins of datasets containing objects with spatial extent. Techniques that keep statistics using irregular space decomposition (e.g., [3]) are not appropriate either, due to the fact that two (or more) joined datasets may not have the same distribution and the space partitions can be totally different. Another limitation of such methods is that information cannot be maintained incrementally, because they need to read the whole dataset in order to update statistics.

Here, we investigate the application of a histogram-based method [28]. We partition the space regularly using a uniform C×C grid and for each cell we keep track of the number of objects and the total length of their MBR per axis. Assuming uniformity at each cell, we can use this information to estimate the selectivity of spatial queries. For example, the selectivity of a range query can be estimated by applying Equation 1 for each cell that is partially covered by the window, summing the results, and adding the number of objects in cells that are totally covered. The selectivity of a pairwise or multiway spatial join [28, 18] can be estimated by applying Equations 2, 3 and 4 (depending on the query) for each combination of cells from the joined datasets that cover the same area, and summing up the results. Figure 6a illustrates a real dataset (T1) that contains road segments from an area in California. Figure 6b presents a regular 50×50 grid that keeps statistical information about the dataset. The z-axis shows the number of objects per cell. Since the characteristics of the dataset vary significantly between cells, application of uniformity-based selectivity formulae is not expected to provide good estimates. The distribution of objects in each cell may not be uniform; however, the skew is not expected to have major effects in the total estimate. This was demonstrated in a previous study [18] where the estimation error for pairwise and multiway joins was within acceptable limits. Moreover, the experimental study in [3] suggests that the accuracy of our method (called *Equi-Area* in this paper) increases significantly with the number of cells. Some histogram-based selectivity estimation methods [22] suggest approximating the distribution of data in a cell by a deviation function instead of assuming uniformity. However, these methods work for (multidimensional) points; the distribution of objects with spatial extents can hardly be described by simple functions. Another important advantage of our method is that statistical information can be maintained incrementally with trivial cost at each insertion/deletion of a rectangle.



(a) dataset T1 (b) number of rectangles per cell in a 50x50 grid

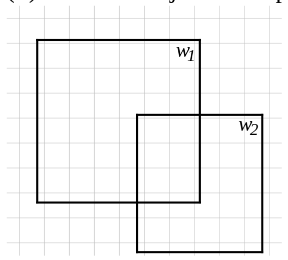
Fig. 6. Skew in dataset T1

In this section we show how the methodology of the previous section for uniform data can be applied to estimate the selectivity of complex spatial queries involving real-life, skewed datasets. We provide formulae that are based on the existence of the 2-dimensional uniform grid and the assumption that rectangles in each cell are uniformly distributed.

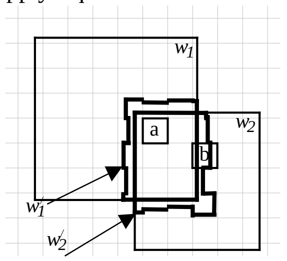
5.1 Selectivity of pairwise joins restricted by selections

We estimate the selectivity of pairwise joins involving skewed datasets that are restricted by selection windows using the methodology described in Section 3.1. Figure 7a shows a configuration of selection windows w_1, w_2 and a statistical grid. We first compute the updated w_1', w_2' for each cell, as illustrated in Figure 7b. Instead of using the global statistics about the average rectangle in a dataset we use the information kept in each cell. Thus the updated windows are not regular rectangles but their length may vary between grids. After the update there might be cells which are totally covered by both windows (e.g., cell *a* in Figure 7b) or partially intersected by them (e.g., cell *b*).

Selectivity is then estimated by summing the join result for each such cell. When a cell is totally covered by both windows, its selectivity is estimated using Equation 2 and considering the area of the cell as the workspace. Otherwise, we apply the methodology described in Section 3.1. Thus we (i) estimate the selectivity of w_1', w_2' in the cell, (ii) estimate the join workspace c and (iii) apply Equation 8.



(a) two windows and a 2D histogram



(b) irregular updated windows using grid information

Fig. 7. Two selection windows and a grid

5.2 Selectivity of multiway joins restricted by selections

As in Section 3.2, we will study two configurations of multiway join queries that are restricted by selections; acyclic and clique (complete) join graphs. The first stage of the estimation involves the computation of the updated windows w'. This is done by applying the window_propagation algorithm of Figure 5. Notice that the updated windows to be considered at each step of the algorithm may be irregular, depending on the rectangle extents at each cell (see Figure 7).

We estimate the output of acyclic queries with selections incrementally using the algorithm of Figure 8. The algorithm first orders the nodes in the query graph, such that each R_i, i>1, in the order is connected to some R_j, j<i. Then at each step i it computes the selectivity of the subquery that includes nodes $\{R_1,R_2,...,R_i\}$ for each cell $g_{x,y}$ of the grid. The number of rectangles that participate in the join at $g_{x,y}$ are estimated by the selectivity of the previous step $OC(g_{x,y},R_1,R_2,...,R_{i-1},w_1',w_2',...,w_{i-1}')$ and the selectivity of w_i', $OC(g_{x,y},R_i,w_i')$. The join result at this step is determined by edge (R_i, R_j), thus, the join workspace $c_{x,y,i,j}$ at cell $g_{x,y}$ is determined by extending w_i', w_j' at both edges and all dimensions by the respective average rectangle extents $\overline{s_{x,y,i,d}}$, $\overline{s_{x,y,j,d}}$ and averaging as described in Section 3.1. The resulting $c_{x,y,i,j}$ is adjusted in all dimensions to be no longer than the respective cell extents, i.e., $\overline{c_{x,y,i,j,d}} = \min\left(\overline{c_{x,y,i,j,d}}, \overline{g_{x,y,d}}\right) \forall\, d$.

Let $OC(g_{x,y},i) = OC(g_{x,y},R_1,R_2,...,R_i,w_1',w_2',...,w_i')$ be the output of the query at step i for cell $g_{x,y}$. In summary, $OC(g_{x,y},i)$ is estimated by multiplying the selectivities of w_i' and the previous sub-query $OC(g_{x,y},i\text{-}1)$ and the join selectivity based on the estimated workspace:

$$OC(g_{x,y},i)=OC(g_{x,y},i\text{-}1)\cdot OC(R_i,\, g_{x,y},\, w_i'\,)\cdot \prod_{d=1}^{k}\min\left(1,\frac{\overline{s_{x,y,i,d}}+\overline{s_{x,y,j,d}}}{\overline{c_{x,y,i,j,d}}}\right) \qquad \textbf{(13)}$$

For clique queries the process is simpler. We assume that the multiway join has a unique workspace which does not vary between join edges, as explained in Section 3.2. Thus, for each dimension we estimate the common intersection of all workspaces c_i and extend it by the average difference of the c_i's from it all each side and dimension. Naturally, like in the previous cases the common workspace might not be a regular window, but we estimate each extent at each cell of the grid. The selectivity of the multiway clique join is then estimated by Equation 12 for each cell after doing the appropriate normalization of the workspace according to the cell's extent at each dimension.

```
selectivity_estimation(window w[], Query Q[][]) {
  window_propagation(w,Q);
  order datasets: $\forall$ i>1, $R_i$ connected to some $R_j$, j<i;
  estimate the selectivity of the first edge ($R_1$, $R_2$);
  for i = 3 to n do {
    let $R_j$ be the node connected to $R_i$, j<i;
    for each cell $g_{x,y}$ of the grid do {
      compute OC($g_{x,y}$, $R_i$, $w_i'$ ); /* OC($R_i$, $w_i'$ )on $g_{x,y}$ */
      compute the join workspace $c_{i,j,x,y}$;
      estimate the join results using Equation 13;
    }
  sum up estimations for each cell and return result;
}
```

Fig. 8. Selectivity estimation for acyclic queries

Table 3. Description of real data used in the experiments

abv.	*Description*	*cardinality (N)*	*density*
T1	California roads	131461	0.05
T2	California rivers and railroads	128971	0.39
G1	German utility network	17790	0.12
G2	German roads	30674	0.08
G3	German railroads	36334	0.07
G4	German hypsography	76999	0.04

5.3 Experimental evaluation

We evaluated the accuracy of the proposed extension of our methodology to handle skewed data by using some real datasets from Geographical Information Systems. The characteristics of the data used in the experiments are provided in Table 3. T1 and T2 are two layers of an area in California with large density differences. The last four datasets capture layers of Germany's map.

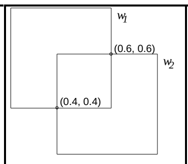
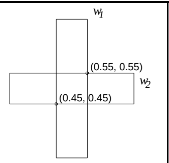
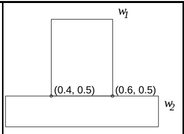
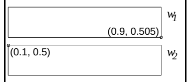
In the first experiment we test the accuracy of our methodology for pairwise joins restricted by selections. We compare the accuracy of Equation 8 which assumes uniformity with the method of Section 4.1 using a 50×50 statistical grid. In Equation 8, instead of the actual average rectangles sides, we used normalized averages taking under consideration that the query workspace is not rectangular, but depends on the area covered by the joined datasets. Table 4 shows the estimates of these methods and the actual query results for various joined pairs and the window configurations of Table 1. The first column for each configuration of windows shows the estimation of Equation 8, the second the estimation of the histograms method and the third the actual result of the query. The results show that both methods are not as accurate as in the case of uniform datasets. Observe that for queries where the windows have some overlap (first and second), applying the grid method is better than assuming uniformity, whereas in queries with trivial window overlap, using histogram information has small effect. This is because the number of results is very small and estimations are more error sensitive.

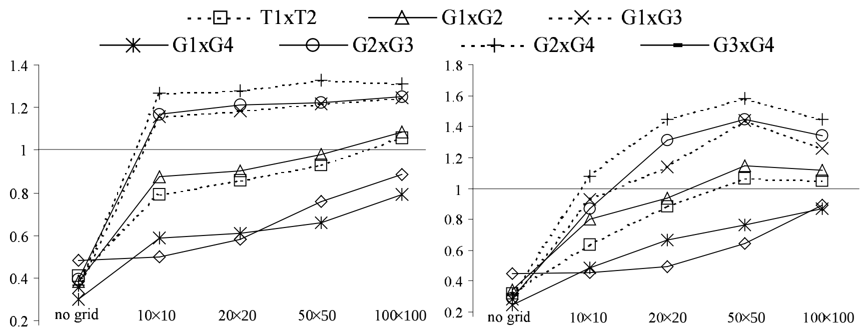
In the next experiment we study how accuracy is affected by the granularity of the grid. For the first two window configurations and various grid sizes we estimated the output of various joins. Figure 9 presents for each join pair and grid size the estimated selectivity divided by the actual query output. Observe that typically the accuracy increases with the detail of the grid, although this is not a rule (see for instance G1 $\bowtie$ G3 in Figure 9a). This is expected, since the more detailed the grid is, the best skew is handled. Nevertheless, very large grids are expensive to store and maintain.

We also studied the accuracy of grids for multiway join queries and the estimates were less precise to the effects of error propagation. Table 5 presents some results when uniformity is assumed and when a 50×50 grid is used. We experimented with two multiway join configurations of the Germany's layers and with the four selection window configurations of Table 2. In general, using the grid is better than assuming uniformity. The estimates are inaccurate for windows with overlap, but the error is not extreme; it is within expected bounds given the increased deviation in pairwise joins and the propagation. On the other hand, in the last two queries where the results are small, error propagation may have large effects (see the last query of the first raw).

We expect windows in typical queries to have some overlap. Thus, the grid can be used without major errors in optimization. On the other hand, when the actual query result is very small, the relative estimation error can be too large. Notice, however, that large relative errors in small results does not affect much estimates in the cost of query operators, since the actual difference translates to few page accesses. The computation cost of the output estimates was negligible. For multiway join queries with selections when the grid is used (the most expensive case) the running time did not exceed few milliseconds, indicating that the estimates can be used for efficient query optimization.

Table 4. Evaluation of the accuracy of the 50×50 grid on pairwise joins with selections

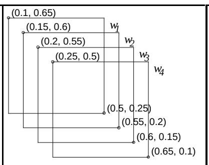
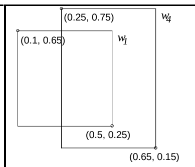
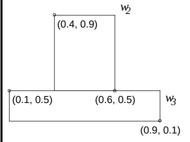
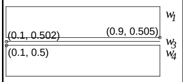
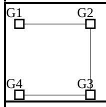
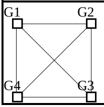
												
	No grid	Grid	Actual	no grid	Grid	actual	no grid	grid	actual	no grid	grid	actual
T1 ⋈ T2	3629	5722	7548	917	1333	2061	27	21	27	0	0	2
G1 ⋈ G2	393	893	958	100	329	309	6	5	3	1	0	2
G1 ⋈ G3	422	1085	1105	107	356	310	6	8	4	2	0	1
G1 ⋈ G4	606	2069	1703	154	778	541	6	16	8	0	0	0
G2 ⋈ G3	407	893	1353	103	319	418	4	4	6	0	0	0
G2 ⋈ G4	505	1573	1284	127	630	436	4	7	6	0	0	0
G3 ⋈ G4	498	1823	1370	125	639	404	3	8	5	0	0	4



(a) grid accuracy: first window configuration (b) grid accuracy: second window configuration

Fig. 9. Accuracy of grids for various joins

Table 5. Evaluation of the accuracy of the 50x50 grid on multiway joins with selections

	 (0.1, 0.65) (0.15, 0.6) (0.2, 0.55) (0.25, 0.5) w_1 w_2 w_3 w_4 (0.5, 0.25) (0.55, 0.2) (0.6, 0.15) (0.65, 0.1)			 (0.25, 0.75) w_4 (0.1, 0.65) w_1 (0.5, 0.25) (0.65, 0.15)			 w_2 (0.4, 0.9) (0.1, 0.5) (0.6, 0.5) w_3 (0.9, 0.1)			 w_1 (0.1, 0.502) (0.9, 0.505) w_3 w_4 (0.1, 0.5)		
	no grid	grid	actual	No grid	grid	actual	no grid	grid	actual	no grid	grid	actual
	68	319	1036	108	650	1856	2	1	0	9	28	0
	15	88	266	25	164	264	0	2	0	0	3	0

6. Discussion

In this paper we have studied the problem of optimizing complex spatial queries that involve multiple spatial joins and selections. We presented formulae that estimate the output of such queries and evaluated them through experimentation. The results prove the accuracy of the formula, the relative error being 8% for binary joins and 38% for queries of four inputs. These numbers are comparable with previous work on selectivity of spatial selections [27] or joins [28], as well as, with error propagation experiments in the context of relational queries [11]. The proposed models are essential for the optimization of queries that involve several spatial logical operators, possibly in addition to some non-spatial ones. We have extended our method for skewed, real-life data using 2D-histograms. In this case, the accuracy is not that high due to the persistence of skew in the cells of the statistical grid, but still the histogram-based method does better than straightforward application of formulae that assume uniformity.

Our contribution is not limited to spatial query processing, since operator dependencies may exist in other applications as well. Consider a relational query that consists of a join between R and S on their common $R.x = S.x$ attribute, and (range) selections on other attributes of R and S (e.g., $R.y < a$, $S.z \in [b,c]$). A dependency between $R.y$ and $S.z$ affects the query results. For instance, in the TPC-R benchmark [26] there are multi-table constraints like O.Orderdate ≤ L.Shipdate (i.e., an order takes place before the corresponding line items are shipped). The selectivity of queries that apply selections on these attributes and then join the corresponding tables can be estimated in a way similar to that presented in this paper (i.e., by restricting the selections, taking under consideration the constraints, and then estimating the join selectivity on the restricted area). Another type of related complex queries involve distance joins of high-dimensional point sets [6]. Our methodology can be applied if high-dimensional selections exist in conjunction with the joins, since the domain of the query operators is the same.

In the future we will investigate additional methods to handle the accuracy problems due to data skew. Another direction for future work is the study of alternative techniques for processing spatial queries. Spatial join operators can be extended to process multiple spatial selections and joins synchronously, since all logical operators in a complex spatial query apply on the same attribute domain. Some preliminary results towards this direction are very promising.

Acknowledgements

Part of this work was conducted while the authors were visiting the Knowledge and Database Systems Laboratory of the National Technical University of Athens, Greece. It was supported by the Research Grants Council of Hong Kong SAR, grants HKUST 6070/00E and HKUST 6090/99E.

References

1. Aboulnaga, A., Naughton, J.F. Accurate Estimation of the Cost of Spatial Selections. *IEEE ICDE*, 2000.
2. Arge, L., Procopiuc, O., Ramaswamy, S., Suel, T., Vitter, J.S. Scalable Sweeping-Based Spatial Join. *VLDB Conference*, 1998.
3. Achaya S., Poosala V., Ramaswamy S. Selectivity Estimation in Spatial Databases. *ACM SIGMOD*, 1999.
4. Belussi A., Faloutsos C. Estimating the Selectivity of Spatial Queries Using the Correlation Fractal Dimension. *VLDB Conference*, 1995.
5. Brinkhoff, T., Kriegel, H.P., Seeger B. Efficient Processing of Spatial Joins Using R-trees. *ACM SIGMOD*, 1993.
6. Faloutsos C., Seeger B., Traina A., Traina C. Spatial Join Selectivity Using Power Laws. *ACM SIGMOD*, 2000.
7. Gaede V., Günther O. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2): 123-169, 1998.
8. Guttman, A. R-trees: A Dynamic Index Structure for Spatial Searching. *ACM SIGMOD*, 1984.
9. Güting R.H. An Introduction to Spatial Database Systems, *VLDB Journal*, 3(4): 357-399, 1994.
10. Graefe, G. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2): 73-170, 1993.
11. Ioannidis Y., Christodoulakis S. On the Propagation of Errors in the Size of Join Results. *ACM SIGMOD* 1991.
12. Ioannidis Y., Kang Y. Randomized Algorithms for Optimizing Large Join Queries. *ACM SIGMOD*, 1990.
13. Kamel I., Faloutsos C. On Packing R-trees. *ACM CIKM*, 1993.
14. Koudas, N., Sevcik, K. Size Separation Spatial Join. *ACM SIGMOD*, 1997.
15. Lo, M-L., Ravishankar, C.V. Spatial Joins Using Seeded Trees. *ACM SIGMOD*, 1994.
16. Lo, M-L., Ravishankar, C.V. Spatial Hash-Joins. *ACM SIGMOD*, 1996.
17. Mackworth A. Consistency in Networks of Relations. *Artificial Intelligence*, 8, 1977.
18. Mamoulis, N, Papadias, D. Integration of Spatial Join Algorithms for Processing Multiple Inputs. *ACM SIGMOD*, 1999.
19. Patel J.M., DeWitt D.J. Partition Based Spatial-Merge Join. *ACM SIGMOD*, 1996.
20. Papadias, D., Mamoulis, N., Delis, V. Approximate Spatio-Temporal Retrieval. To appear in *ACM Transactions on Information Systems*.
21. Papadias, D., Mamoulis, N., Theodoridis, Y. Processing and Optimization of Multiway Spatial Joins Using R-trees. *ACM PODS*, 1999.
22. Poosala, V. Histogram-based estimation techniques in databases. PhD Thesis, *University of Wisconsin-Madison*, 1997.
23. Pagel, B.W., Six, H. Are Window Queries Representative for Arbitrary Range Queries? *ACM PODS*, 1996.
24. Pagel, B.W., Six, H., Toben, H., Widmayer, P. Towards an Analysis of Range Query Performance. *ACM PODS*, 1993.
25. Silberschatz, A., Korth, H.F., Sudarshan, S. *Database System Concepts*. McGraw-Hill, 1997.
26. Transaction Processing Performance Council, TPC Benchmark R (Decision Support), Rev. 1.0.1, http://www.tpc.org/, 1993 – 1998.

27. Theodoridis, Y., Sellis, T. A Model for the Prediction of R-tree Performance, *ACM PODS*, 1996.
28. Theodoridis, Y., Stefanakis, E., Sellis, T. Cost Models for Join Queries in Spatial Databases, *IEEE ICDE*, 1998.
29. Tsang, E. *Foundations of Constraint Satisfaction*, Academic Press, London and San Diego, 1993.