VIII Brazilian Symposium on GeoInformatics, Campos do Jordão, Brazil, November 19-22, 2006, INPE, p. 73-90.

A Rule-Based Optimizer for Spatial Join Algorithms

Miguel Fornari[1,3], João Luiz Dihl Comba[1], Cirano Iochpe[1,2]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – CEP 91.501-970 – Porto Alegre – RS – Brazil

[2]PROCEMPA - Empresa de Tecnologia da Informação e Comunicação da Prefeitura de Porto Alegre – Av. Ipiranga, 1200, CEP 90160-091 – Porto Alegre – RS – Brazil

[3]FATEC-SENAC – Faculdade de Tecnologia-SENAC/RS – Rua Cel. Genuíno, 130 – CEP 90010-030 - Porto Alegre – RS – Brazil

fornari@ieee.org, {comba, ciochpe}@inf.ufrgs.br

Abstract. *The spatial join operation is both one of the most important and expensive operations in Geographic Database Management Systems (GDBMS). This paper presents a set of rules to optimize the performance of the filtering step of spatial joins operations. First, a set of expressions to predict the number of I/O operations and CPU performance is presented. The rules are based on expressions to predict the performance of algorithms and tests performed with synthetic and real data sets. Fos some cases, the optimized algorithm can execute the same operation 10 times faster than the original, non-optimized version.*

1. Introduction

The spatial join operation combines two sets of spatial features, *A* and *B*, based on a spatial predicate [Rigaux et all, 2000]. Combining such pairs of spatial features in large data sets implies the execution of both Input/Output (I/O) and a large number of CPU operations. Therefore, it is both one of the most important and the most expensive operations in geographic databases systems (GDBMS).

As an example, consider a geographical data set describing rivers and another one representing counties. Many applications, like transport planning, agricultural production and flood prevention, must know which counties are crossed by rivers. To answer the query "find every county that is spatially crossed by a river" a user can apply a spatial join over the two feature sets with the topological predicate "crosses".

Traditionally, a user submits a query to the Database Management System (DBMS), using a high level language, such as SQL. After lexical and syntactic validation, the query is transformed into a relational algebra expression, to be processed by the query optimizer module. The query optimizer, based on a set of statistical data stored in the data dictionary, defines an execution plan. The evaluation engine performs the query according to the execution plan over the user data.

The main contribution of this work consist of a set of rules to optimize the performance of some well-known algorithms: *Partition Based Spatial Merge Join (PBSM)* [Patel and De Witt, 1996], *Iterative Stripped Spatial Join (ISSJ)*, Synchronized

Tree Transversal (STT) [Brinkhoff et all, 1993] and *Histogram-based Hash Stripped Join (HHSJ)* [Fornari and Iochpe, 2004].

The goal is to reduce the response time of spatial join algorithms, changing some basic parameters. We must address two main problems: (1) *which parameters are relevant for the algorithm´s performance* and (2) *what is the best value for each important parameter?*

The text is structured as follows. In the section 2 the expressions to predict the number of I/O operations and CPU performance are introduced. The section 3 explains the system architecture of the software implemented to carry out the tests using real and synthetic data sets. The set of rules are described and justified in the fourth section. Section 5 presents some conclusions and suggestions.

2. Spatial Join Algorithms

The algorithms that perform the filtering step of the spatial join operation manipulate object descriptors, defined by their object identifier (OID), the minimum bounding rectangle (MBR) and a pointer to the geometric description of the object.

Figure 1 shows a complete set of algorithms that perform the filtering step of a spatial join operation. The algorithms are classified according to the file organization used to maintain object descriptors. As mentioned before, we select a representative algorithm for each class, except for the nested loops and one-indexed file classes. In preliminary tests, nested loops presented very long response time, being the worst choice in any situation. The algorithms in the class "one indexed file" are an extension of the "pure" algorithms for a special case, and we expected that the results can be extended for them.

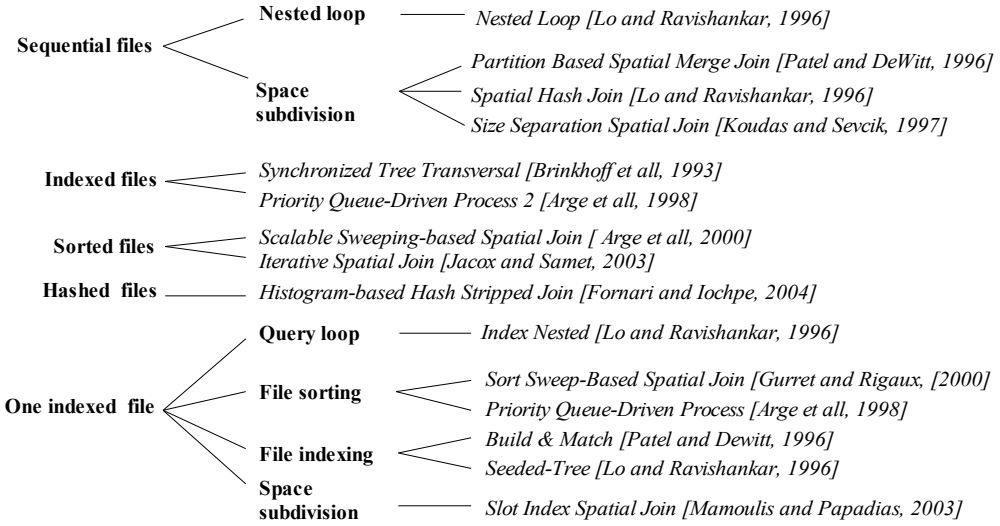

Figure 1. Classification of spatial join filter step algorithms according to their method of file organization.

2.1 Plane-sweep technique

All spatial join algorithms, in different ways, load objects into memory and a kind of plane-sweep strategy to check if object pairs satisfy the spatial predicate[Rigaux et al., 2000][De Berg, 2000].

The plane-sweep technique has a performance of $O(k+n\log n)$, where k represents the number of intersections between objects and n is the number of objects [De Berg, 2000]. If the technique is used to find intersections between objects of two sets, then $n=n_A+n_B$, where n_A and n_B is, respectively, the number of objects in sets *A* and *B*. The value of k can vary a lot, depending on the spatial distribution and the size of objects.

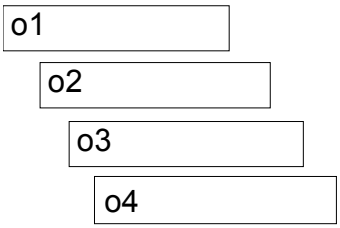

Figure 2. An example of a set of objects where *k=0* and *c=6*.

In order to predict the algorithm performance in the most accurate way, we define the number of pairs for which the spatial predicated is verified, expressed by c, independently if the result is true or false. Figure 2 shows an example where the number of intersections (k) is zero, but the number of comparisons (c) is six, because the pairs (o_2,o_1), (o_3,o_2), (o_3,o_1), (o_4,o_3), (o_4,o_2) and (o_4,o_1) are tested.

In a first approach, considering the objects uniformly distributed in the space, c can be estimated by

$$c=s_x n \tag{1}$$

where s_x, represent the average size of objects. This technique also works if the objects are sorted by their y coordinates. The estimation in this case is $c=s_y n$.

For non-uniform distributions, we divide the space into strips, as can be seen in figure 3. For each strip, the number of overlapping objects is counted, and a object distribution histogram is built.[Belussi et all, 2003][Belussi et all, 2004]. The estimation can be made by the following expression, where Φ is the number of strips and n_i is the number of objects of sets *A* and *B* in strip i, for $1 \leqslant i \leqslant \Phi$.

$$c=\sum_{1}^{\Phi} n_i \tag{2}$$

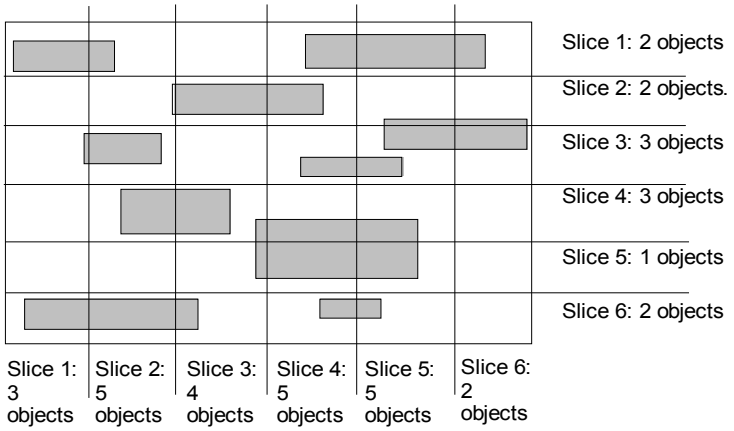

Figure 3. An example of space stripping.

Although the size of objects might not appear to be important, it has an impact in the number of objects in each strip. Figure 3 shows an example of a set of objects in a space divided by 6 strips. As can be observed, the shape of objects significantly modifies the number of objects per slice.

Table 1. The symbols and their meanings.

Symbol	Meaning
k	Number of intersecting pairs of objects
c	Number of pairs of objects tested to verify if attends to the spatial predicate
n^A, n^B	Number of objects in sets *A* or *B*, respectively
$s_x^A, s_y^A, s_x^B, s_y^B$	Average size on axle *x* or *y*, for sets *A* or set *B*.
b^A, b^B	Number of disk blocks occupied by sets *A* or *B*.
M_o, M_b	Memory size on number of objects and number of blocks
t_{sr}, t_{sw}	Average time to sequential read/write operations.
t_{rr}, t_{rw}	Average time to random read/write operations.
t_{coord}	Time to compare two coordinates, in double precision
t_{inter}	Time to verify if two objects intersects
t_{RPM}	Time to verify if two objects intersects and if attends the *Reference Point Method* (RPM).
h, h_A, h_B	Height of R-Trees.
n_i^A, n_i^B	Number of nodes on *i* level of a R-Tree.
$s_{x,i}^A, s_{y,i}^A,$ $s_{x,i}^B, s_{y,i}^B$	Average size of nodes in level i of a R-Tree.
Fanout	Fanout of a R-Tree
$s_x^{Strips}, s_y^{Strips}$	Average size on strips that divides the space.
Φ, Φ_H, Φ_V	Number of slices that divides the space
P	Number of partitions
ρ	Probability of a certain block to be in the memory buffer .
o	Percentage of partitions that overflowed
b_o	Number of overflow buffers
r	Replication factor

The plane-sweep technique is adapted according to the spatial join algorithm. These modifications can alter the value of *c*, and are described within the respective algorithm. Table 1 summarizes the symbol notation used in this paper.

2.2 Synchronized Tree Transversal

The Synchronized Tree Transversal (STT) [Brinkhoff et all, 1993] algorithm needs that both sets are indexed, in advance, for two different R-Trees, named R_A and R_B. Nodes of both R-Trees are compared, in a synchronized way. The number of nodes comparisons was defined in [Huang e Jing, 1997], as

$$Comp = \sum_{i=1}^{h-1} \sum_{1}^{n_i^A} \sum_{1}^{n_i^B} (min(s_{x,i}^A + s_{x,i}^B), 1) \times (min(s_{y,i}^A + s_{y,i}^B), 1) \quad (3)$$

Initially, the height of both trees are considered to be equal and is expressed just by *h*. Be n_i^A the number of nodes of set *A* in level *i* of the R-Tree R_A, and $1 \leqslant i \leqslant h$. The average size, in the axis *x*, of nodes or MBR in level *i* is represented by $s_{x,i}^A$.

Based on [Theodoridis, 1996][Theodoridis, 1998][Huang,1997], the number of I/O operations when the buffer size is equal to zero is defined as $Zj = 2 + 2Comp$. Introducing a buffer, ρ represents the possibility of a certain node being found in buffer memory, reducing the number of I/O operations to

$$io_{STT} = n_R^A + n_R^B + (Zj - n_R^A - n_R^B)\rho \quad (4)$$

The expression for the CPU performance depends on the number of nodes comparisons (3). For each node comparison, *2Fanout* objects are involved, at maximum, resulting in

$$cpu_{STT} = O(c_{STT} + 2\, Fanout \log 2\, Fanout) \quad (5)$$

An adequate value for c_{STT} is necessary to complete the expression. The R-Tree divides the space in an irregular way. Two objects, aligned on the *x* axis, can be allocated in different nodes. As a result, they will not be compared to evaluate the spatial predicate, reducing *c*. If let s_1^R be the average size of leaves, *c* can be estimated by

$$c_{STT,1} = \frac{c}{s_1^R} \quad (6)$$

The value of *c*, for internal levels, depends on the number and average size of nodes of the lower level.

$$c_{STT,i} = min(s_{x,i+1}^A + s_{x,i+1}^B, 1) \times n_{i+1}^A \times n_{i+1}^B \quad (7)$$

Add the number of comparisons in all levels, the value of *c* for the entire R-Tree is obtained.

$$c_{STT} = \sum_{i=1}^{h} c_{STT,i} \quad (8)$$

2.3 Iterative Stripped Spatial Join

The Iterative Stripped Spatial Join (ISSJ) algorithm is an adaptation of the Iterative Spatial Join, proposed by Jacox and Samet [2003]. The main idea is, first, to sort the sets of objects separately. Then, the plane-sweep technique is applied, scanning both

sets in sequence. During the plane-sweep, just the active objects are maintained in memory. The space is divided in strips to reduce the number of comparisons. When an object is loaded to memory, it is assigned to one or more strips. A different active objects list is maintained for each strip. In this way, pairs of distant objects are not compared.

Considering a uniform distribution of objects, in each strip the number of objects is $r(n^A+n^B)/\Phi$, where r represents the replication factor. It can be calculated as the number of objects, with replicas, over the original number of objects, $r=(n_R^A+n_R^B)/(n^A+n^B)$, resulting in a value greater than 1.

The number of I/O operations depends on whether the sorting is internal or external. The best case occurs if both sets are sorted using an internal algorithm. Then, the number of I/O operations is

$$io_{ISSJ}=3(b^A+b^B) \tag{9}$$

The worst case occurs if both sets are sorted by an external algorithm. The number of I/O operations increases to

$$io_{ISSJ}=(b^A+b^B)+2b^A\times\lceil\log_{M_b}(b^A)\rceil+2b^B\times\lceil\log_{M_b}(b^B)\rceil . \tag{10}$$

Due to the space division in the strips, the number of pair comparisons is reduced to

$$c_{ISSJ}=\frac{r\,c}{\Phi} \tag{11}$$

The expected performance of the algorithm is

$$cpu_{ISSS}=O(c_{ISSJ}+r(n^A+n^B)\log\frac{(r(n^A+n^B))}{\Phi}) . \tag{12}$$

2.4 Partition Based Spatial Method

As a first step, the PBSM [Patel and DeWitt, 1996] divides the space into a set of cells by applying a regular grid to it. A partition is a set of cells and each cell belongs to one, and only one, partition. A spatial object may intersect one or more adjacent cells, belonging to different partitions. The algorithm replicates the object descriptor in all intersecting partitions.

The objects are loaded to memory, distributed in partitions, and then reloaded to memory to the plane-sweep. For each object, three I/O operations are executed. But, a certain percentage of partitions, represented by *o*, overflows, because all objects don´t fit in memory at the same time, forcing another read/write operation. Replicas in the result set are avoided using the *Reference Point Method* (RPM) [Dittrich and Seeger, 2000]. Thus, the number of I/O operations can be expressed by:

$$io_{PBSM}=(2or^2+2r+1)(b^A+b^B) . \tag{13}$$

Due to space subdivision, the number of pairs of objects comparisons is reduced. If the number of horizontal cells is greater than the number of partitions, as figure 4a

shows, in a same row, more than one cell can be allocated to the same partition. If the number of horizontal cells is smaller, as in figure 4.b, considering just one row, each cell is allocated to a different partition. Being the number of horizontal cells represented by Φ_H, the value for *c* can be estimated by

$$c_{PBSM} = \frac{c}{\Phi_H},\ P > \Phi_H$$

$$c_{PBSM} = \frac{c}{P},\ P \leq \Phi_H \quad . \tag{14}$$

1	5	4	3
2	1	5	4
3	2	1	5
4	3	2	1

(a)

1	2	3	1
2	3	1	2
3	1	2	3
1	2	3	1

(b)

Figure 4. Cell allocation, according to the number of partitions, using a round-robin schema.

In a uniform distribution, all partitions have the same number of objects, including replicas. In this case, the plane-sweep technique for each partition processes $r(n^A + n^B)/P$ objects. As the plane-sweep is repeated for all partitions, the total number of comparisons is expected to be in an order of

$$cpu_{PBSM} = O(c_{PBSM} + r(n^A + n^B)\log\frac{r(n^A + n^B)}{P}) \quad . \tag{15}$$

2.5 Histogram-based Hash Stripped Join

The Histogram-based Hash Stripped Join (HHSJ) [Fornari and Iochpe, 2004] has three main characteristics: the object descriptors are stored in a hash file organization; a bi dimensional histogram of object distribution defines the spatial extension of each partition; and, when it loads objects to memory, it divides the space by strips.

The histogram is maintained in a quadtree, so-called *HistQ*. Each of its quadrants represents a subdivision (a cell) of the global space. Each node contains a counter of the number of objects that intersects the space segment represented by this node of the quadtree. Each level of the quadtree represents a different histogram of object distribution, with a different degree of precision.

A hash file is created for each data set. It is organized in buckets, assuming a static hash organization. The hash function associates a leaf node of *HistQ* to a specific bucket in the hash file. Therefore, there exists one bucket in the hash file for each leaf of the quadtree. If an object transposes quadrants and is counted in two (or more) leaf nodes of *HistQ*, its descriptor is replicated in two (or more) buckets. The number of replicated objects is defined during the creation of the hash file.

The minimum number of buckets is defined by

$$Buckets = \lceil \frac{n}{Fanout} \rceil . \quad (16)$$

Using (16), the height of the *HistQ* can be calculated by

$$h = \lceil \log_4 Buckets \rceil . \quad (17)$$

In fact, the number of buckets, and, considering a disk block for each bucket, is

$$b = h^4 . \quad (18)$$

Because some buckets in the hash file can overflow, some additional disk blocks are necessary.

To carry out the test of spatial predicate in pairs of objects, first, the HHSJ loads the *HistQ* of both sets to main memory. Based on them, partitions and respective boundaries are defined. Each partition is spatially defined by the corresponding selected quadrants in both quadtrees. The space is divided in an irregular way. Regions more densely populated are divided into small areas, to maintain the number of objects pertaining to a partition into the available memory.

Before, the algorithm processes each partition separately. First it identifies the buckets that are covered by the partition. Then, the object descriptors stored in each bucket are loaded to memory.

The algorithm just loads to memory both *HistQs* and all buckets, including overflow buckets, of each hash file. So, the number of I/O operations is

$$io_{HSSJ} = b_A^{HistQ} + 4^{h_A} + bo_A^{Hash} + b_B^{HistQ} + 4^{h_B} + bo_B^{Hash} \quad (19)$$

The number of comparisons between pairs of objects is reduced by two factors: the number of partitions (P) and the number of strips inside each partition (l). As the space is divided in an irregular way, we use the mean partition size, expressed by s_x^{Part} and a constant number of strips, for all partitions, resulting in

$$c_{HHSJ} = \frac{c}{s_x^{Part} \times \Phi} . \quad (20)$$

the performance in CPU can be estimated by

$$cpu_{HHSJ} = O(c_{HHSJ} + r(n^A + n^B) \log \frac{r(n^A + n^B)}{p\Phi}) . \quad (21)$$

3. The System Architecture

The performance analysis, although correct, simplifies many cases. To compare the algorithms in real situations, a software system was implemented to carry out the tests between them. The main components of its architecture are shown in figure 5.

To acquire real data sets, one tool converts different data formats to the internal files. Another tool generates synthetic data sets. This second type of data is valuable for performance tests, because the user can control the parameters that describe the data set

characteristics, changing one or two of them in each data set, and then test the algorithms to verify differences in the their performance.

The main component in the system architecture is the join module, which contains the spatial join algorithms implemented. The design of the join module allows one to plug and play other algorithms easily.

When running a test, the user specifies the desired algorithm and the size of main memory. The buffer capacity is measured in number of pages and each page has a fixed size of 4Kb. For the STT algorithm, a specific R-Tree Oriented Buffer was implemented for better performance. The other three algorithms use a traditional LRU buffer.

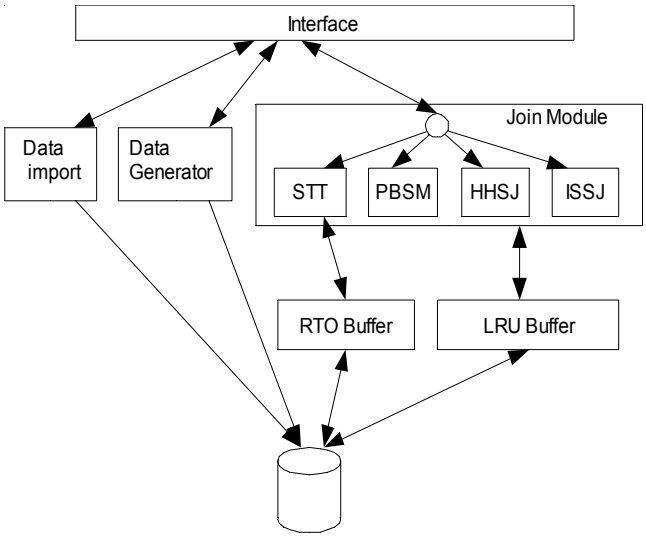

Figure 5. The main component in the implemented system architecture is the join module, containing a set of different algorithms.

For all algorithms, the overall response time and number of I/O operations can be obtained. For specific algorithms, other values, like the replication factor or the size and height of R*-Tree, can be obtained.

The entire system was implemented in C language, for Linux operating system, by us. During tests, the system operating buffer was turned off, so as to not influence the result time. All tests were performed on an Intel 2.4GHz, with 512 Mb of RAM and SCSI disks.

3.1 Data Sets

The artificial data sets were generated by the system, and the real data sets were obtained from different sources. Table II shows the name, description, cardinality and average size on both axes for each real data set, grouped by its source. As can be seen, we used a great number of real data sets, with very different characteristics. The cardinality has an obvious importance, because it defines the workload for the algorithm. The average size of objects is important because it defines, to a great extent, the number of pairs of objects that will be compared.

Table 2. Real Data sets Characteristics

Name	Description	Cardinality	# of disk blocks	% of average size on X	% of average size on Y
Source: R-Tree Portal (www.rtreeportal.org)					
ca_streets	Californian streets - multilines	2.249.727	13.157	0,016%	0,013%
ca_streams	Californian streams of water - multilines	98.451	576	0,184%	0,157%
ge_roads	German roads - multilines	30.674	180	0,146%	0,108%
ge_utility	German public utility networks - multilines	17.791	103	0,236%	0,169%
ge_rrlines	German railroads - multilines	36.334	213	0,117%	0,084%
ge_hypsogr	German hypsographic data - multilines	76.999	451	0,067%	0,048%
gr_roads	Greek roads - multilines	23.278	137	0,188%	0,187%
gr_rivers	Greek rivers - multilines	24.650	145	0,187%	0,200%
la_streets	Los Angeles streets - multilines	131.461	769	0,035%	0,029%
la_rr	Los Angeles railroads - multilines	128.971	755	0,086%	0,071%
Source:Bureau of Transportation Statistics (USA) – www.bts.gov					
usa_counties	Counties - polygons	3.236	19	0,0049%	0.01%
usa_rr	Railroads - multilines	166.688	981	0,00049%	0,0041%
usa_hydro	Rivers - multilines	517.538	2959	0,00049%	0,0009%

4. Rules for Performance Optimization

In this section, the set of rules for performance optimization is presented. Each rule is explained and exemplified, showing the results of performed tests.

Rule 1: the DBMS can estimate *k* for each axle and choose the one with minor value of *k,* optimizing the plane-sweep.

The rule proposes that the GDBMS, first, must estimate the number of comparisons to be made in each axle, and, after, performed the spatial join algorithm sorting the data by the choosen axle. Two estimation approaches are suggested, one based on a single formula (1), another based on histograms (2).

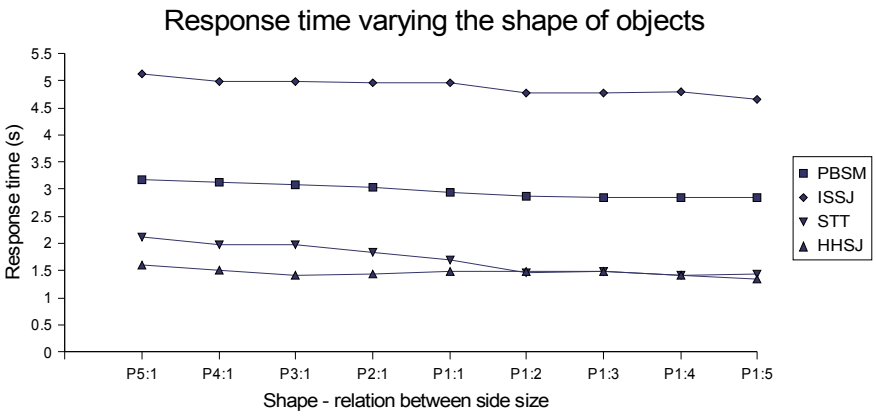

Figure 6. Graph of response time varying the shape of objects.

Firstly, we generate a number of synthetic data sets, varying the shape of the MBR, from a relation of 5:1 to a relation of 1:5. The cardinality of data sets are, always, 200.000 objects. The graph in figure 6 shows the response time for each algorithm, sorting by the same axle in all tests, making clear the minimizing possibilities of the axle choice.

We run all possible joins between real sets , sorting by both axles to validate both alternatives. When the difference between response time was less than 5%, we discard the join, because the difference is closed that both sorting alternatives seems to be good enough. In this way, we concentrate in greater differences, where the choice are relevant. Alternative 1, based just in mean size, choose the right axle in 29 times, but the wrong axle 10 times, because it is a rough simplification of the data sets, representing a 74.3% of correct match. Alternative 2 presented a 100% of correct match, when the space is divided in 500 strips. The number of correct match is not affected by the spatial join algorithm.

This result indicates clearly that the histogram based prediction is superior and must be used always that is possible, justifying the necessity to maintain the histograms always that an object is inserted or deleted from the data set. But, in a query plan, the spatial join can be performed after another operation, like a selection. In this case, the histograms are not available. Our suggestion is, during the anterior operation, create the histogram, according to the temporary set are written.

Rule 2: The STT algorithm is optimized defining nodes with a low number of entries. The total number of nodes will be greater, elevating the value of *Comp*, and defining a minimum limit for the rule.

The rule 2 intend to optimize the STT algorithm. The construction of R*-Trees is performed using the STR algorithm [Leutenegger et all, 1997] to reduce the number of nodes and obtain a maximum occupation of entries in each node. This method reduces the number of I/O operations, because the size of R*-Trees are minimized. Second, the buffer algorithm try to maintain non-leaf nodes in the memory buffer, because non-leaf nodes are more susceptible to reloads than leaf nodes, also, reducing the number of I/O operations.

The rule changes the number of entries in a node, called *fanout*. The number of leaf nodes can be calculated as $n_{Leafs} = \lceil n / fanout \rceil$. The number of non-leaf nodes, is proportional to the number of leaf nodes. So, if the fanout is reduced, the number of nodes are incremented, maybe, adding a new level to the R*-Tree. By expression (3), the number of node comparisons are, also, increased, limiting the rule 2. Figure 7 shows the response time for three different cases of spatial join, varying the fanout from 10 to 146, that is the maximum fanout for a node maintained in just one block.

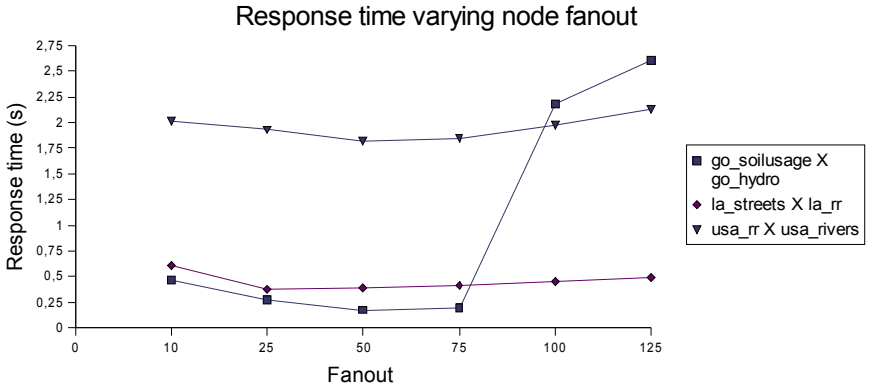

Figure 7. Graph of response time using the STT algorithm, varying the fanout.

Also, the influence of the buffer memory size was verified. The performance of STT algorithm is constant when the memory size increases, confirming the results showed by [Gatti and Magalhães, 2000]. In fact, the time to execute I/O operations represents a small part, in general, less than 2% of the total time, although the buffer hit ratio increases according to the memory size, reducing the number of I/O operation performed, as expected.

Rule 3: The ISSJ algorithm is optimized definining a great number of strips. The number of objects in each strip will be small, but the is limited by the adding of replicas.

The third rule is to increase the number of strips to optimize the algorithm ISSJ. But, this rule increments the replication factor, which penalizes the performance.

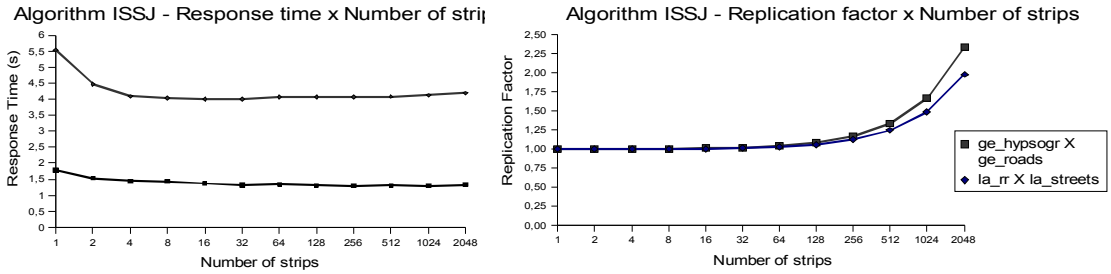

Figure 8. Graph of response time and replication factor, for the ISSJ, varying the number of strips.

The graph in figure 8 shows the response time of two different spatial joins, varying the number of strips and the replication factor. As can be seen, a small number of strips, four to eight, presents an increment in algorithm performance, reducing the response time. After this number of strips, the response time almost stabilize or present

a small increment, because the number of processed objects (original + replicas) increases exponentially.

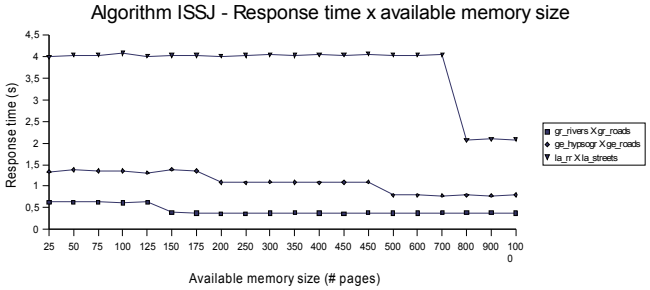

Figure 9. Graph of response time for algorithm ISSJ, varying the available memory size.

Changing the available memory size, the performance of the algorithm suffers great impact of changing the sorting algorithm. The number of I/O operations is reduced, as well as, CPU cost. In the graph of figure 9, this fact is clear, with a step down according to the increase of memory size, in the exact point where there is enough memory to do an in-memory sorting of one of the joined sets. All operations were performed with a constant number of 8 strips.

Rule 4: The PBSM algorithm is improved setting a high value for the number of partitions using a small size of memory or just set a lower bound to the number of partitions. This rule is limited by the number of replicas, that increase the number of processed objects.

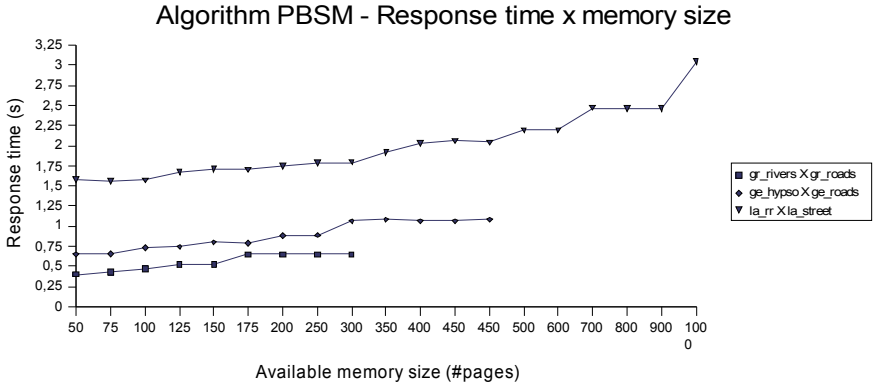

Figure 10. Graph of response time for the algorithm PBSM, varying the available memory size.

The rule 4 increases the performance of the PBSM algorithm by incrementing the number of partitions. In this rule, the replication factor is, also, increased, impacting the performance. The GDBMS can allocate less memory to the algorithm to control the number of partitions or directly set a value for *P*. Figure 10 shows the graph of response time of three spatial join operations, for which the available memory size was changed, to induce different number of partitions. As the memory size increases, the number of partitions is reduced, in general, increasing the response time, as expected from (15). In all cases, the algorithm uses a 32x32 grid of cells.

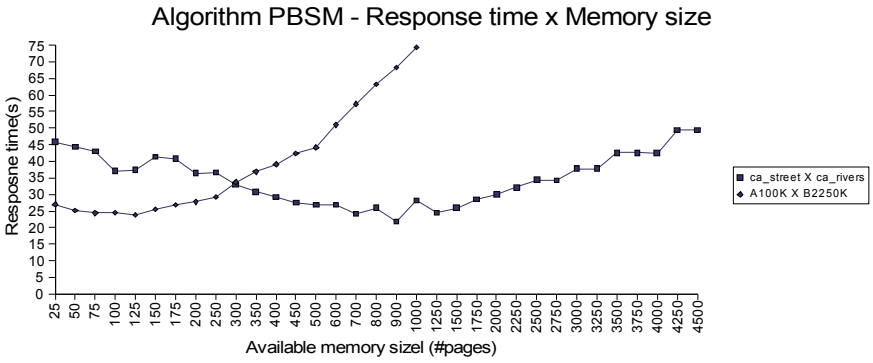

Figure 11. Graph of response time for two cases joined using PBSM algorithm, varying the available memory size.

Two cases can be seen in figure 11: one between real sets *ca_streets* and *ca_streams;* another between two synthetic data sets. They have the same cardinality of the real sets, but objects are uniformly distributed in the space.

The set *ca_streets*, with more than 2 millions of objects, is the key to understand the behaviour of response time. In small memory size, the number of partitions is greater than the number of cells defined by the default grid of 32x32. A redefinition of the grid to 64x64 cells is necessary, to map at least one cell for each partition. Using a finer granularity grid, the replication factor increases. Also, mapping just a few cells to one partition, more concentrated areas result in partitions that overflow, being necessary to execute a costly repartition procedure. As the available memory size increases, the distribution of objects in partitions become uniform, avoiding repartitions. After a certain memory size, the response time increases.

Rule 5: The HHSJ is improved setting a large value for the number of partitions and for the number of strips. This rule is limited, also, by the number of replicas, that increase the number of processed objects.

The rule 5 is a combination of the strategies 3 and 4, as the HHSJ algorithm combines aspects of both the ISSJ and the PBSM algorithms. Replication can occur in two moments when HHSJ is executed: during the creation of the hash file; and when objects are loaded into memory and divided by strips. As the creation of the hash file can be done only once when the set is loaded to the GDBMS, we concentrate our attention in the second moment, that occurs always when the set is involved in a spatial join operation.

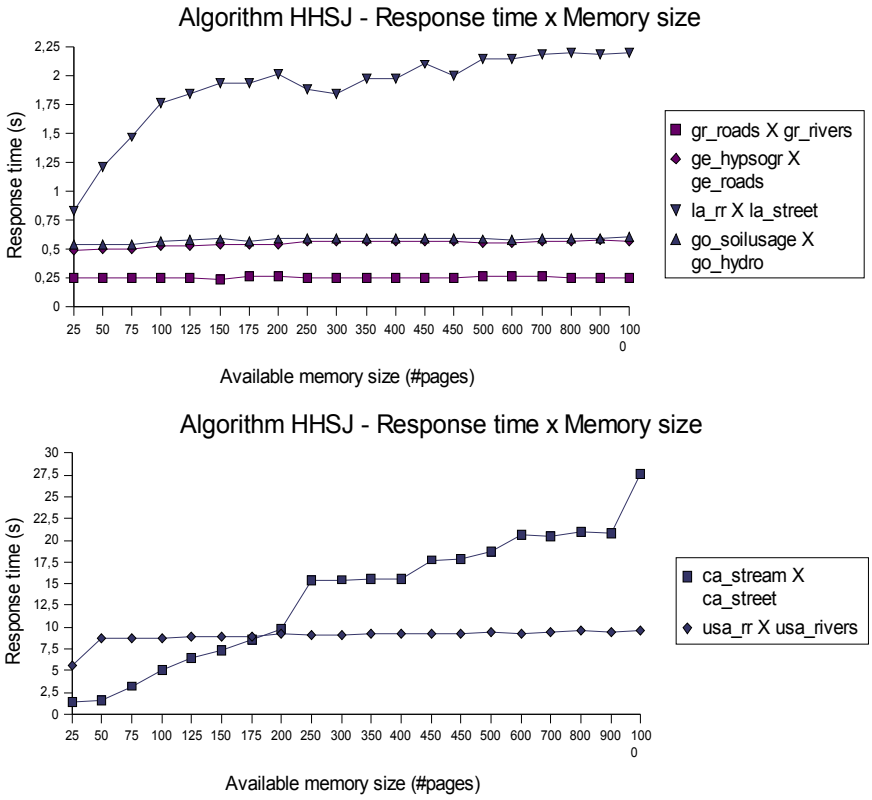

Figure 12. Two graphs of response time for the algorithm HHSJ, varying the available memory size.

Figure 12 shows two graphs of response time against available memory size. The number of partitions varies according to memory size, but we kept in 16 the number of strips in all cases. For small sets, the response time is almost constant. For larger sets, the response time increases with the available memory size, being constant after a certain amount of memory. This behavior is different than PBSM, where the response time just increase, not stabilizing. In HSSJ, the creation of strips, in memory, allows the stabilization, establishing a maximum value for the response time. For the largest set spatial join operation, *ca_street* and *ca_stream*, the stabilizing point is achieved just at almost 8Mb of memory, and is not visible in the graph.

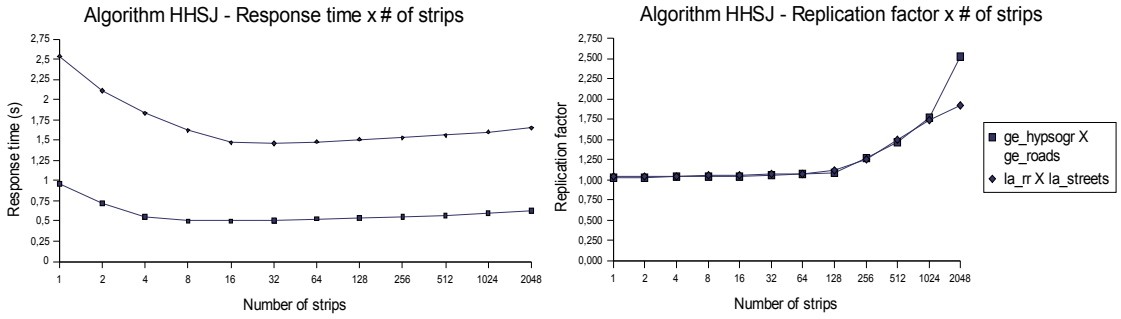

Figure 13. Two graphs of response time and replication factor for the algorithm HHSJ, varying the number of strips.

The graph in figure 13 shows the effect of change the number of strips in algorithm HHSJ, maintaining the available memory size. The behavior of response time

is almost the same of the ISSJ: a small number of strips results in an important decrease of response time; a great number of strips produces many replicas, increasing the response time.

If the available memory size increases, the number of I/O operations is not changed, because the algorithm read all buckets only once to perform the operation.

5. Conclusions

This paper first introduce expressions to predict the number of I/O operations and CPU performance of each studied algorithm, using an unified notation. After, a set of simple rules are established to improve the performance of such algorithms. Using tests with real and synthetic, response times were collected to prove the correctness of rules. In some situations, for example, spatial joining the sets *ca_streets* and *ca_stream* using PBSM, the response set varies between 20.3s and 48s, a difference of more than 50%. In some extreme cases, like optimizing the fanout in R-Trees, the STT algorithms can perform the spatial join more than 10 times faster.

The study concentrates in spatial join operation using bi-dimensional sets, but the proposed strategies can be easily extend to operate in three or more dimensional spaces and spatiotemporal data structures. Although tests with real data sets permit the evaluation of very different scenarios, we do not expected to cover all possibilities. The experiments can be carried out for other system architectures like palmtops, where some additional constraints can impact the performance. Another possibility is running in a grid architecture, but this imply a significative change in the code of investigated algorithms.

The goal of reducing CPU time can be explored in many other areas, like statistic databases, clustering spatial objects and text mining algorithms. Another aspect to investigate is the implications of CPU caches with 2Mb or more, in traditional databases algorithms.

Also, we propose that available GDBMS, like Oracle Spatial [Oracle, 2003], IBM DB2 Spatial Extender [IBM, 2005] and PostGis [Refractions Inc., 2005] can incorporate the proposed rules and more alternative file organizations, like sorted and hash files, and algorithms, like ISSJ and HHSJ. In general, they implement only some variation of R*-Trees, reducing the number of spatial join algorithms to just three: STT, if both sets are spatial indexed; Scan&Index, if one set is indexed; and nested loops, if there is no indexes. But, even for this algorithms, some rules can increase the performance when executing spatial joins, because the differences in response time are large.

References

Arge, L., O. Procopiuc, O. et al. (2000) “A Unified Approach for Indexed and Non-Indexed Spatial Joins”. Proc. of 7th Int´l. Conf. on Extending Database Technology, p. 413-429, 2000.

Arge, L., Procopiuc, O. et al. (1998) "Scalable Sweeping-Based Spatial Join" Proc. of VLDB, 1998, p.570—581, 1998.

Belussi, A.; Bertino, E. and Nucita, A. (2003), "Grid Based Methods for Spatial Join Estimation" Proc. of 11TH Symp. of Advanced Databases Systems, p.49-60, 2003.

Belussi, A.; Bertino, E. and Nucita, A.(2004). "Grid Based Methods for Estimating Spatial Join Selectivity" Proc. of 12th ACM GIS, 2004.

Brinkhoff, T.; Kriegel, H.P. and Seeger, B. (1993), "Efficient processing of spatial joins using R-trees" Proc. of ACM SIGMOD, p. 237-246, 1993.

Dittrich, J.P. and Seeger, B. (2000), "Data Redundancy and Duplicate Detection in Spatial Join Processing" Proc. of Int´l. Conf. on Data Engineering, p. 535-543, 2000.

Gatti, S.D. and Magalhães, G.C. (2000), "A Comparison Among Different Syncronized Tree Transversal Algorithms for Spatial Joins" Proc. of GeoInfo, 2000 - available in www.geoinfo.info.

Gurret, C. and Rigaux, P. (2000) "The Sort/Sweep Algorithm: A New Method for R-Tree Based Spatial Joins" Proc. of Statistical and Scientific Database Management, p.153-165, 2000.

Huang, Y.W. and Jing, N. (1997), "A Cost Model for Estimating the Performance of Spatial Joins Using R-Trees" Proc. of Int´l Conf. on Information and Knowledge Management, 1997.

IBM (2005) , "IBM DB2 Spatial Extender- User´s Guide and Reference Version 8", 2005.

Jacox, E.H. and Samet, H. (2003), "Iterative Spatial Join" ACM Transactions Database Systems, v.28, n.3, p. 230—256, 2003.

Koudas, N. and Sevcik, K.C.. (1997) "Size Separation Spatial Join", Proc. of ACM SIGMOD, p. 324-335, 1997.

Leutenegger, S. T.; Edgington, J. M.; Lopez, M.A. (1997) "STR: A Simple and Efficient Algorithm for R-Tree Packing". Technical Report- TR-97-14. University of Denver, Mathematical and Computer Science Department, Denver: USA.

M-L. Lo and C.V. Ravishankar (1996), "Spatial hash-joins" Proc. of ACM SIGMOD, 1996, p. 247—258, 1996.

M. De Berg, M, M. Van Kreveld, M. Overmars and O. Schwarzkopf (2000), "Computational Geometry", 2nd. edition. Springer-Verlag, 2000.

M.R. Fornari and C. Iochpe (2004), "A Spatial Hash Join Algorithm Suited for Small Buffer Size", Proc. of ACM GIS, p. 118-126, 2004.

Mamoulis, N. and Papadias, D. (2003), "Slot Index Spatial Join" IEEE Trans. on Knowledge and Data Engineering, v.15, n.1, p. 211-231, 2003.

ORACLE, "Oracle Spatial User´s Guide and Reference 10g Release 1", Dec., 2003.

Patel, J.M. and DeWitt, D.J. (1996), "Partition Based Spatial-Merge Join" Proc. of ACM SIGMOD, p. 259-270, 1996

REFRACTIONS RESEARCH (2005), "PostGIS Manual." Disponível em http://postgis.refractions.net/docs/postgis.pdf.

Rigaux, P.; Scholl, M. and Voisard, A. (2000), "Spatial Databases with Applications to GIS". Morgan Kaufmann Pub., 2000.

Theodoridis, Y.; Sellis, T. (1996) "A Model for the Prediction of R-Tree Performance". Proc. of 16th. ACM Symposium on Principles of Database Systems, ACM Press, 1996, p.161-171.

Theodoridis, Y.; Stefanakis, E.; Sellis, T. "An Efficient Cost Model for Spatial Joins Using R-Trees" IEEE Transactions On Knowledge And Data Engineering,v.12, n.1 , p.19-32, January,2000.