A comparison among different synchronized tree traversal algorthims for spatial joins

Sandro Danilo Gatti[1], Geovane Cayres Magalhães[1],

[1] Institute of Computing, Univ. of Campinas - Caixa Postal 6176, 13083-970, Campinas, SP, Brasil
{gatti,geovane}@dcc.unicamp.br

Abstract. Join techniques for spatial access methods were analysed. The factors considered included buffer pool size, page size and intermediate join indexes ordering criteria. This analysis was based on real data taken from a GIS applications for telecommunications. Results of this work assess the way those factors affect spatial join performance and can be used for tuning such algorithms.

1 Introduction

We all have seen the great advances of informatics. New and different applications arise every day, followed by the development of faster machines and sofisticated software. In this context, we have geographic applications applied in areas such as precision agriculture, urban planning, utlities, and so on. These applications are assisted by Geographic Information Systems (GIS), which deals with spatial data.

Spatial data have characteristics that make them different from conventional data. That urges the development and tuning of algorithms that manage and use these data efficiently. Among these algorithms and operations, spatial joins can be included.

This work aims to evaluate some spatial join proposals taken into consideration on buffer pool and page sizes, as well as intermediate join indexes ordering criteria, which affect enormously spatial joins performance. It takes the R*-tree as the base index technique.

This paper is organized as follows: section 2 presents works that influence this investigation; section 3 presents a description of the system used in the tests, the analysis criteria and the data used; the section 4 presents the results obtained in this work, as well as some discussions; and finally, the section 5 presents the conclusions of this work.

2 Related works

2.1 Spatial access methods

Many spatial access methods have been proposed and can be classified on different ways [1, 2]. This works takes as its base the R*-tree [3], an evolution of Guttman's R-trees [4].

R-tree are indexing structures derived from B-trees. Within the R-trees, the space is hierarchically divided into regions, with the lower levels regions being enclosed by higher ones. Regions are modelled by minimum bounding rectangles (MBR). Basically, R-trees have two kinds of nodes:

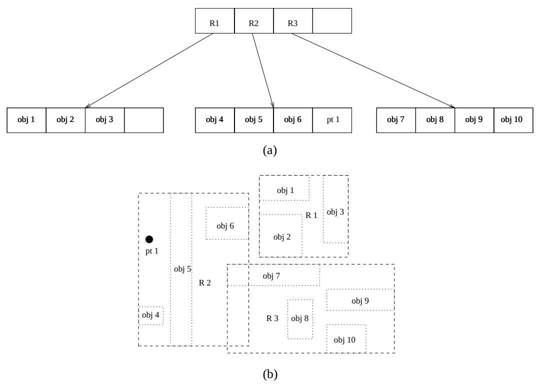


Figure 1: An R-tree (a) and the indexed points and rectangles (b).

- leaf nodes, which store information about the indexed objects in the form (*id*, *MBR*), where the first field is the object identifier and the second one is the object MBR;
- non-leaf nodes, which contains information on the hierarchy (*pointer, MBR*), where *pointer* is a pointer to a node of a lower level and *MBR* is and MBR that encloses the MBR of the children node.

Every node is stored in a disk page, which maximum capacity is M entries like (*pointer, MBR*) or (*id, MBR*). The minimum number of entries is $m \leq \frac{M}{2}$.

Figure 1 shows the points and rectangles indexed by an R-tree. Note that the MBRs that belong to the same level can intersect. This structure is used by the conventional R-tree and by the R*-tree. The difference between R-trees and R*-tree remains in the overflow and insertion treatment algorithms, which, in the R*-tree implementation, try to decrease the MBRs overlapping and coverage areas.

2.2 Spatial join methods

After the develpment of spatial access methods, research efforts turned to the develpment of more efficient algorithms for insertion, deletion and retrieval operations. Retrieval (query) is the most important operation, and in this class, spatial joins deserve special attention.

Spatial joins are an important class of spatial query. This kind of query is used to answer questions like "Which public buildings are close to squares?" or "Which roads cross rivers ?".

These operations are analogous to relational joins, that use conventional data and also demand large computational effort. But, instead of dealing with conventional data, spatial joins deal with spatial data and spatial predicates, such as "intersection", "inclusion", among others.

Relational joins frequently uses the equality criteria as the comparison predicate, while spatial join hardly uses equality, using intersection among objects instead.

The predicates used on spatial joins hinders the use of conventional joins algorithims. This is initially due to the fact that spatial data do not have a total natural ordering which preserves spatial locality. Besides, conventional joins algorithms are optimized for equality. But spatial joins hardly ever use equality, which makes it difficult the use of conventional algorithms.

The researches in this area resulted in some methods, optimized for use when there are not indexes available on the entries, when only one entry is indexed, and when both entries are indexed.

When there are not indexes available, it is possible to apply a technique that uses partitioning, assigning tuples to buckets and using a plane-sweeping technique. Methods that use this approach are the PBSM (*Partition Based Spatial-Merge Join*) [5], the SSSJ (*Scalable Sweeping-Based Spatial Join*) [6], the *Spatial Hash-Join* [7] and another variant of this one, introduced by Zimbrão in [8].

If only one of the sets is indexed, it's possible to use an scan-and-index approach, or to build at run-time an index on the non-indexed set, and then run the join. Proposals which attack this problem are the *seeded-tree*, presented in [9], and the *SISJ* (*slot index spatial join*), presented in [10].

Finally, if both sets are indexed, a general join method can be applied, tuning it to the chosen access method. A grid-based access method could use some hashing technique and a tree-based access method could use a synchronized tree traversal (STT) technique [11, 12, 13].

Spatial joins, as well as other spatial queries, can be run in two steps: *filtering*, when candidates to the answer set are chosen and *refinement*, when the real shape of the object is retrieved and evaluated against the predicate. The approaches that should be paid attention to, when dealing with two indexed sets, are the STT methods.

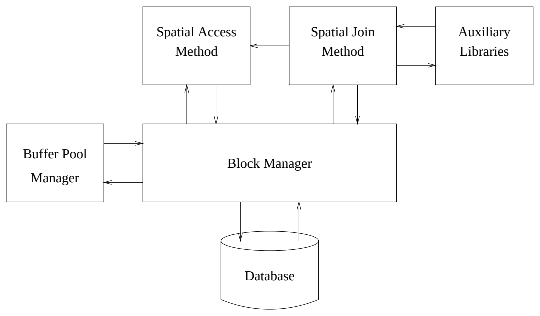


Figure 2: The system architecture.

3 Implementation

This work emphasizes time requirements, given in terms of I/O operations. The join methods chosen for implementation are the ones based on STT, presented in the works presented in [11], [12] and [13].

3.1 System architeture

A modularized system was implemented in order to ease the evaluation, which gives total control of the operation to the researcher. It is formed by the disk abstraction module (DAM), the spatial join module (SJM), the index module (IM) and the auxiliary libraries modules (ALM). These modules were implemented in the C language and the tests were run on a SUN SparcStation 20, with 128 megabytes of memory.

The index and disk abstraction modules where reused, with some modifications, from previous works [2, 14]. The system architecture is shown in Figure 2.

The DAM is the responsible for the interface between the system and the SJM, the IM and the operating system. It also controls a buffer pool and parameters such as buffer pool size, page size, amount of pages to be pinned in the buffer pool and page replacement policies. The IM keeps the indexing and data handling. Currently, our work uses R*-trees, but can also incorporate other indexing methods.

The SJM includes three spatial join methods: the *nested loop* (NL) [11], a *depth-first* (DF) [12], and a *breadth-first* (BF) [13]. The ALM keeps auxiliary functions, such as sorting, list manipulation, and other operations.

3.2 The data

The data used in our work is a real data set, obtained from SAGRE Project (Sistema Automatizado de Gerência de Rede Externa) [15, 16, 17], developed by the Centro de Pesquisa e Desenvolvimento (CPqD). This system automatizes processes related to management, planning, designing, expansion, among other operations, of an outside telecommunication network. The data sets are representative to ap-

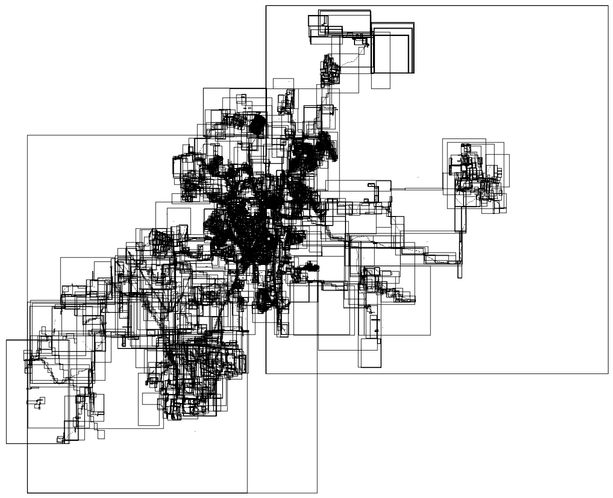


Figure 3: MBRs referring to *city* set.

plications such as power distribuition facilities, telephony, water distribution facilites, and others.

These data sets are constituted basically by a sequence of double float elements. Every four doubles form a M-BR in the form $(x_{min}, y_{min}, x_{max}, y_{max})$. These data are kept in three different sets: the first contains points, named *poles*, the second contains rectangles, named *blocks* and the third contains points and rectangles, named *city*. Figure 3 shows a graphical representation of the *city* data set.

The *poles* set contains 13,813 MBRs, the *blocks* set contains 2,473 MBRs and the *city* contains 66.837 MBRs, which includes the *poles* and *blocks* sets and also elements such as manholes, cables, pipelines, and others.

The data sets have non-uniform distribution and sizes. The *city* data set have a high density. These sets were indexed into three distinct R*-trees, one for the *poles* set, other for the *blocks* set and another for the *city* set. For each set, we have trees with the following page sizes: *1k*, *2k*, *4k* e *8 kbytes*.

3.3 Tested queries

All the queries were set to run a join between the *poles* and the *blocks* sets, which page sizes varying among $1k$, $2k$, $4k$ e *8 kbytes*. Buffer pool sized varied among 2, 4, 8, 16, 24 e 32 pages. Different ordering criteria were used to sort the intermediate join indexes (IJI): an ordering by z curve (SZ), and the ordering obtained by the plane-sweeping technique for the DF method; the ordering obtained by the plane-sweeping technique (NS), sorting by x_{min} of one of the entries (SO), sorting by the sum of the x centers of the interction MBRs (SS), sorting by the x center of the MBR that encloses the intersecting MBRs (SC), and an ordering by a Hilbert curve (SH).

Every query was run 10 times, computing the average CPU time, from the same initial conditions. The buffer replacement policy was LRU (least recently used).

Other tests were also done, running a join between *blocks* $\times$ *blocks* and *city* $\times$ *city*. For these last queries, the buffer pool size varied from 2 to 256 disk pages.

4 Discussion and results

The results of this work are presented now, divided into CPU results, memory usage, and I/O results. Except when explicitly mentioned, all the results were obtained from the *poles* $\times$ *blocks* join.

4.1 CPU time and main memory usage

CPU time is not our main interest. But some measures were done in order to check the influence of the IJI ordering criteria and disk page size in spatial join performance. Figure 4 (a) shows a comparison between the three methods, not considering ordering times, while Figures 4 (b) and (c) shows the results for the DF and BF methods with sorting, respectively.

From the experiments related to CPU, we can see that all the methods are influenced by page size. This is due to the fact that, with smaller pages, the tree is higher and the nodes MBRs enclose a smaller area, containing less elements.

It's also possible to see that *plane-sweeping* techniques contribute to reduce de CPU time spent by the BF and DF methods, compared to the NL method. On the other hand, the use of sorting increased the CPU time, mainly in the BF method, which has to deal with large IJIs.

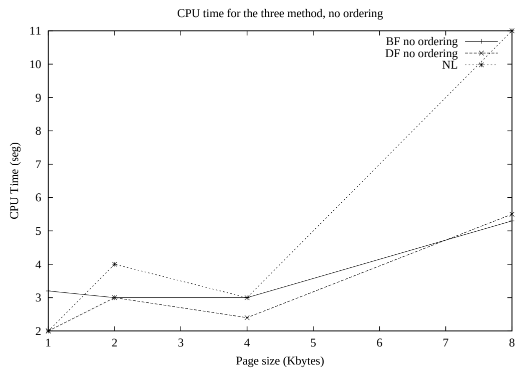
It is important to say that the BF method used much more memory than the other methods in order to store the IJIs, mainly when used 1 kbyte and 4 kbytes page. This also justifies the peaks shown for the BF curves for 1 kbyte and 4 kbytes pages, the page sizes for which the IJIs had the largest sizes.

4.2 I/O operations

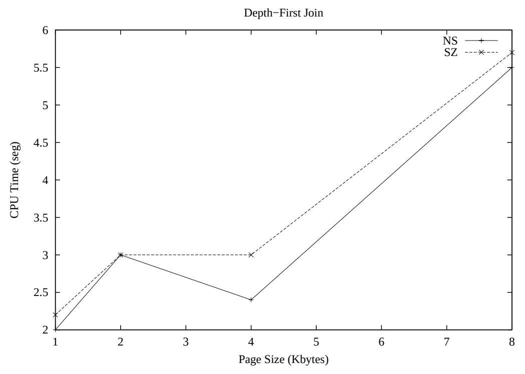
We present now the results for the I/O operations obtained by the NL, DF and BF methods. The NL method was the worst one when running *poles* $\times$ *blocks*. But it's useful in order to compare the gains we may have with some policies.

4.2.1 Results for DF method

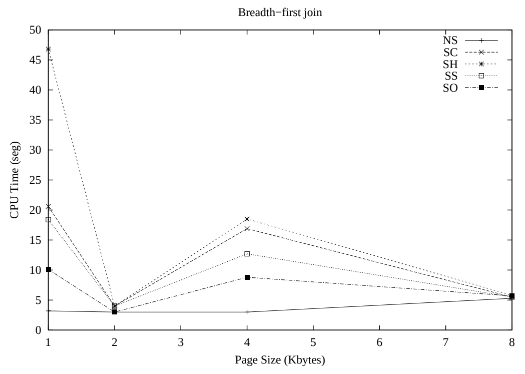
The Figure 5 shows the results for the DF method, comparing with the results of the NL method. Although not shown here, the results for 1 kbyte and 2 kbytes pages reflect these results. It's possible to see that the improvements introduced resulted into better performance.



(a) CPU time not considering ordering time

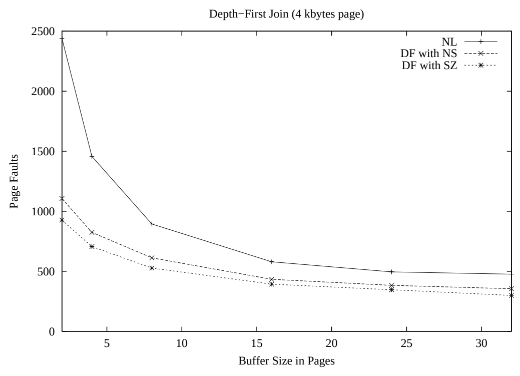


(b) CPU time for DF method

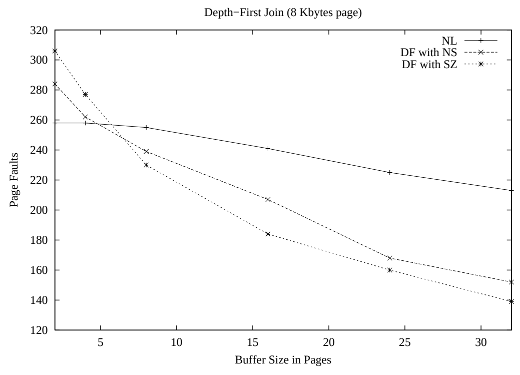


(c) CPU time for BF method

Figure 4: CPU time spent by the different methods and a comparison among them.

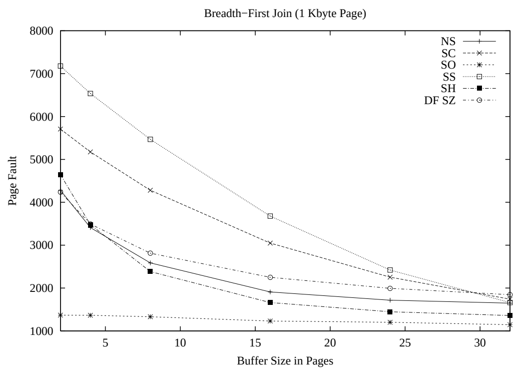


(a) 4 kbyte page

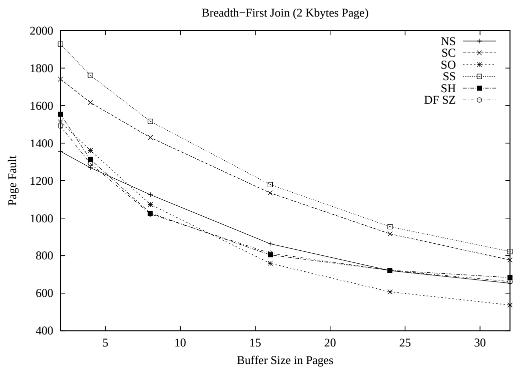


(b) 8 kbyte page

Figure 5: Results for the DF method, using indexes on *4 kbytes* (a) e *8 kbytes* (b).



(a) 1 kbyte page



(b) 2 kbyte page

Figure 6: Results BF method *1 kbytes* (a) e *2 kbytes* (b) pages.

But, opposite to Brinkhoff *et al.* [12], we believe that it's possible the use of the z curve in order to sort the IJI. In their work, Brinkhoff *et al.* said that the gain in terms of I/O operations did not paid the extra CPU expenses when sorting by z curve. But today we have faster machines and capable of handling the extra workload of z curve sorting by, so that the gap between the expenses of NS and SZ would be very small or vanish.

4.2.2 Results BF method

The BF method, more than the DF method, is highly influenced by the order the pairs are place in the IJI. The results of the BF tests for *1 kbyte* and *2 kbytes* can be seen in Figure 6, comparing to DF method with SZ.

The best ordering criteria for the BF method was SO, mainly when we have indexes with different heights. In opposit to the DF method, the BF method was able to read all the needed pages just once. This is due to this method nature: it's possible to know all the pages that are necessary to the join in advance, allowing a better ordering of the IJI. On the other hand, this method demanded much more memory than the DF method. And, according to the size of this IJI, it would be necessary to store parts of it in the disk, what causes more accesses to disk and could nullify the benefits of this method. Just to have an idea of how big the IJI can be, some experiments generated IJIs with 70.7% of the index size.

4.2.3 Other results

In addition to the experiments shown tests, others were run. These tests were *blocks* $\times$ *blocks* (BB) and *city* $\times$ *city* (CC). The results for BB tests are according to the ones already presented here. On the other hand, the CC for DF tests diverged from these ones.

With CC tests, the NL method was superior to the DF method, considering I/O results. The BF method was still the best concerning this criteria, but when the buffer pool was relatively small, NL was better.

These results can be caused by the difference of sizes and distribution of the *city* set. Some analysis were done and they show that there are some MBRs that cover great part of data space (Figure 3). These large MBRs results into large intermediate MBRs, which spread to higher level, increasing the overlapping among internal node. This is directly related to the data insertion routine in the index: if the insertion routine generates internal levels with high overlap, it will affect performance of all the queries. But, in order to know, without doubts, the real cause of this anomaly, other experiments should be performed, with data of several configurations, what will be the target for new tests.

5 Conclusions

The results of these test lead us to conclude that:

- the increase of page size results the increase of CPU workload. So, the increase of a page size should be followed by improvement of the plane-sweeping algorithms;

- a good ordering of the IJI is very important to the good performance of the join. In our tests, the best were SZ for DF and SO for BF;

- a good performance of spatial joins also depends on a buffer pool with a good proportion of pages. In our tests, a buffer with good proportion was about 10% or more of the size of the indexes in pages;

- the use of the BF algorithm can, depending on the data density, turn into prohibitive IJI sizes;

- there are data sets where the use of the DF method results into worse performance, in number os I/O operations, than the NL method;

- insertion and manipulation routines have fundamental importance on all kinds of queries.

In short, spatial join methods could and should be improved, taking into account the factors analysed here. The gains that the tuning of these factors can bring justify completely this work and new researches.

Acknowledgements

We would like to thank CPqD for the data sets used in this work. Thanks also to CNPq for the financial support of this work, through Pronex/SAI project.

References

[1] B. C. Ooi. *Efficient query processing in geographic information systems*, volume 471 of *Lecture Notes in Computer Science*. Springer-Verlag Inc., New York, NY, USA, 1990.

[2] F. S. Cox and G. C. Magalhães. Implementação e análise de métodos de acesso adados espaciais. In *VII Simpósio Brasileiro de Banco de Dados*, Porto Alegre, Maio 1992.

[3] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceeding of the 1990 ACM SIGMOD International Conference on Management of Data*, Junho 1990.

[4] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1984.

[5] J. Patel and D. DeWitt. Partition based spatial-merge join. In *SIGMOD*, pages 259 – 270, Montreal, Canadá, Junho 1996.

[6] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. Vitter. Scalable sweeping-based spatial join. In *Proc. of the 24th VLDB Conference*, New York, USA, 1998.

[7] M. Lo and C. V. Ravishankar. Spatial hash-joins. In *Proceedings of the 1996 ACM SIGMOD*, SIGMOD RECORD, pages 247–258, Montreal, Canadá, Junho 1996. ACM Press.

[8] G. Z. da Silva. *Avaliação de Junções em Bancos de Dados Espaciais*. PhD thesis, COPPE/UFRJ, Rio de Janeiro, Junho 1999.

[9] M. Lo and C. V. Ravishankar. Spatial joins using seeded trees. In *Proceedings of the 1994 ACM SIGMOD*, volume 23 of *SIGMOD RECORD*, pages 209–220. ACM Press, Junho 1994.

[10] N. Mamoulis and D. Papadias. Integration of spatial join algorithms for processing multiple inputs. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, Filadéfia – Pensilvânia, Junho 1999.

[11] O. Günther. Efficient computation of spatial joins. In *International Conference on Data Engineering*, pages 50–60, Los Alamitos, Ca., USA, Abril 1993. IEEE Computer Society Press.

[12] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using R-trees. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 22(2):237–246, junho 1993.

[13] Y.-W. Huang, N. J., and E. A. Rundensteiner. Spatial joins using R-trees: Breadth-first traversal with global optimizations. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*, pages 396–405, 1997.

[14] A. P. Carneiro. Análise de desempenho de métodos de acesso espaciais baseada em um banco de dados real. Master's thesis, Universidade Estadual de Campinas - UNICAMP, 1998.

[15] G. C. Magalhães. Telecommunications outside plant managements throughout Brazil. In *Conference XX Proceendings*, Nashville, 1997.

[16] G. C. Magalhães. Projeto SAGRE. *Fator GIS*, Out./Nov./Dez. 1993.

[17] G. C. Magalhães. The development of open systems for engineering applications. In *Proc. of XVII Intl. Conference on AM/FM*, Denver - Colorado, Março 1994.